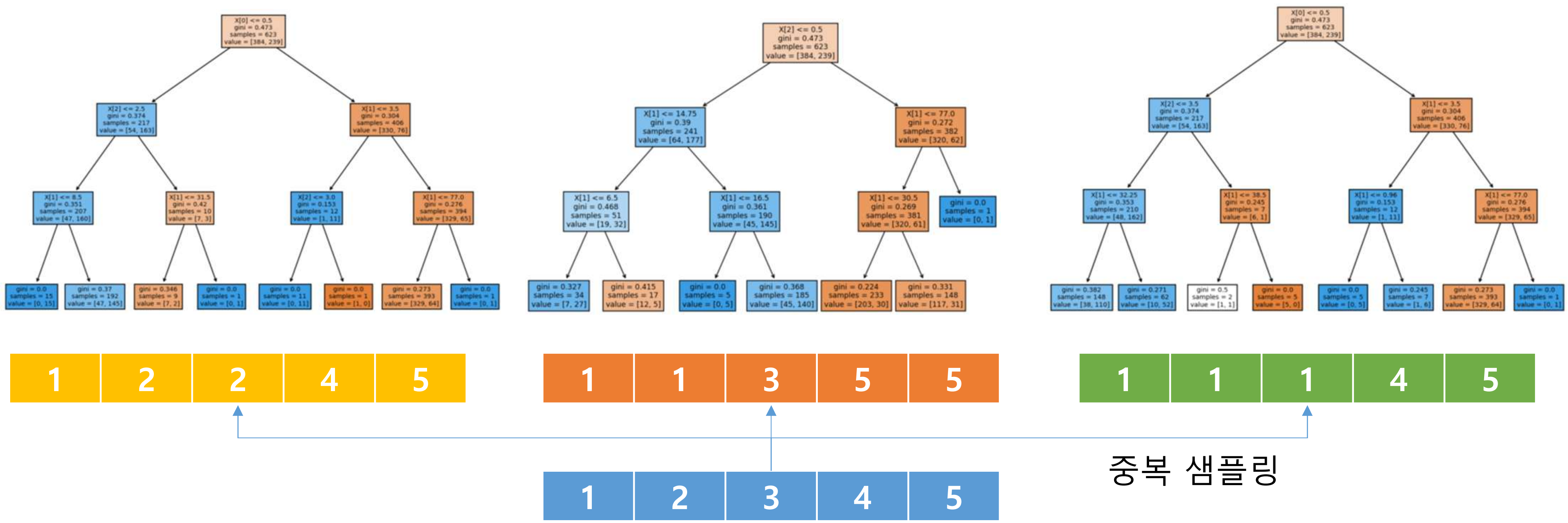
배깅(Bagging)

Bagging(Bootstrap+Aggregating)

- Bootstrapping: 전체 데이터로부터 여러 개의 데이터 세트를 중첩되게 무작위로 샘플링하는 방식
- Aggregating: 여러 모델을 사용하여 얻은 결과들을 취합하여 최종 클래스 결정



의사결정나무 분류

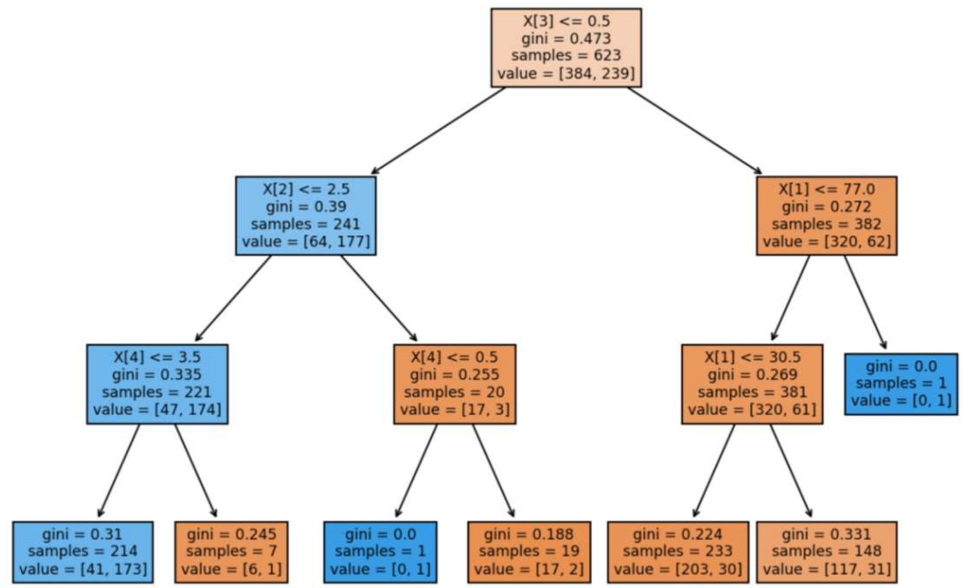
```python
import seaborn as sns
titanic = sns.load_dataset("titanic")
data = titanic[["sex", "age", "sibsp", "adult_male", "parch"]].copy()
t = titanic["survived"]
data["age"].fillna(30, inplace = True)
data["sex"].replace("male", 1, inplace = True)
data["sex"].replace("female", 0, inplace = True)

from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)
```

```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth = 3, random_state = 42)
```

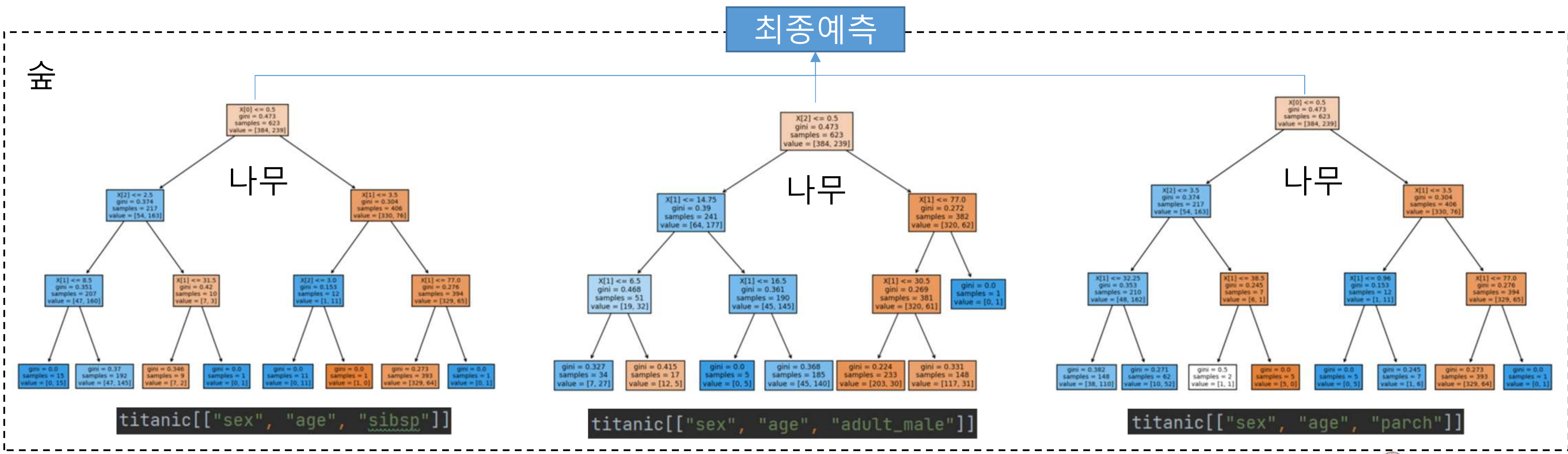
```python
model.fit(train_data, train_target)
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

```
Train-Eval: 0.8314606741573034
Test-Eval : 0.8134328358208955
```



랜덤포레스트(Random Forest)

- 여러 분류기들의 결과를 결합하여 최종 예측을 수행하는 앙상블(Ensemble) 학습 사용
- 일부 특징을 무작위로 선택하여 여러 개의 의사결정나무를 만들고 숲을 구성한 뒤, 숲을 통해 최종 예측
- 의사결정나무 각각의 결정을 다수결로 최종 판단하거나, 확률을 평균하여 최종 클래스 예측



sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

n_estimators : *int, default=100*

The number of trees in the forest.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

`feature_importances_` : ***ndarray of shape (n_features,)***

The impurity-based feature importances.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

랜덤포레스트 분류

```python
import seaborn as sns
titanic = sns.load_dataset("titanic")
data = titanic[["sex", "age", "sibsp", "adult_male", "parch"]].copy()
t = titanic["survived"]
data["age"].fillna(30, inplace = True)
data["sex"].replace("male", 1, inplace = True)
data["sex"].replace("female", 0, inplace = True)

from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)
```

```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state = 42)
model.fit(train_data, train_target)
```

```python
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

```
Train-Eval: 0.8956661316211878
Test-Eval : 0.7723880597014925
```

```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state = 42)
model.fit(train_data, train_target)
```

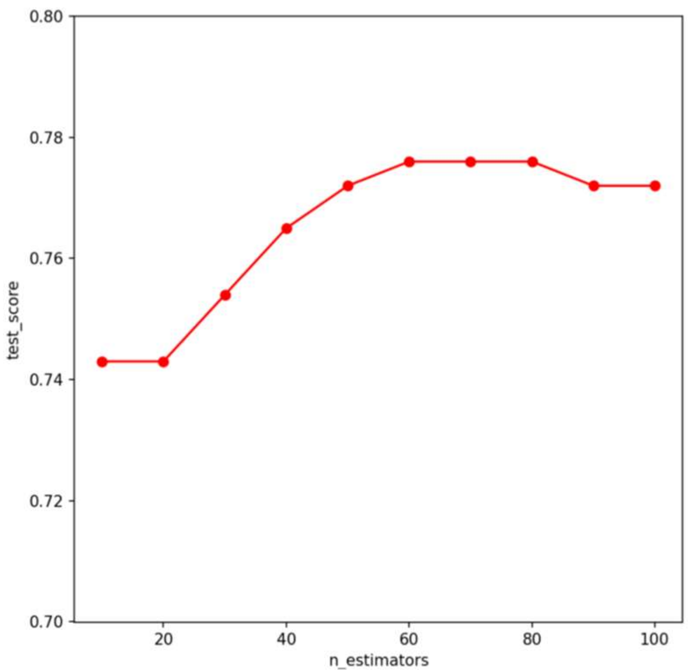
```python
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

```
Train-Eval: 0.8956661316211878
Test-Eval : 0.753731343283582
```

의사결정나무 수

n_estimators : *int, default=100*
The number of trees in the forest.

```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10, random_state = 42)
model.fit(train_data, train_target)
```



n_estimators	train_score	test_score
10	0.889	0.743
20	0.891	0.743
30	0.889	0.754
40	0.892	0.765
50	**0.896**	0.772
60	0.894	**0.776**
70	**0.896**	**0.776**
80	**0.896**	**0.776**
90	**0.896**	0.772
100	**0.896**	0.772

그리드서치(max_depth)

```
from sklearn.ensemble import RandomForestClassifier
```

구현
(max_depth={3, 5, 7, 10}, 10-폴드, random_state=42)

```
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

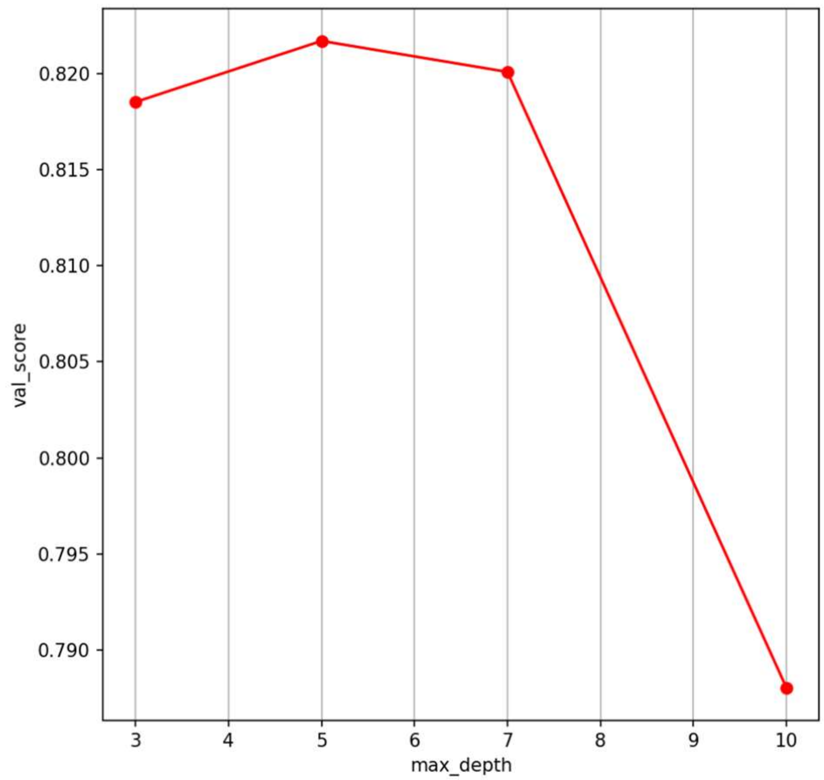
```
Train-Eval: 0.8314606741573034
Test-Eval : 0.8097014925373134
```

```
import matplotlib.pyplot as plt
```

구현

```
plt.show()
```

```
[0.3041013  0.14438944 0.09452992 0.38597277 0.07100658]
```



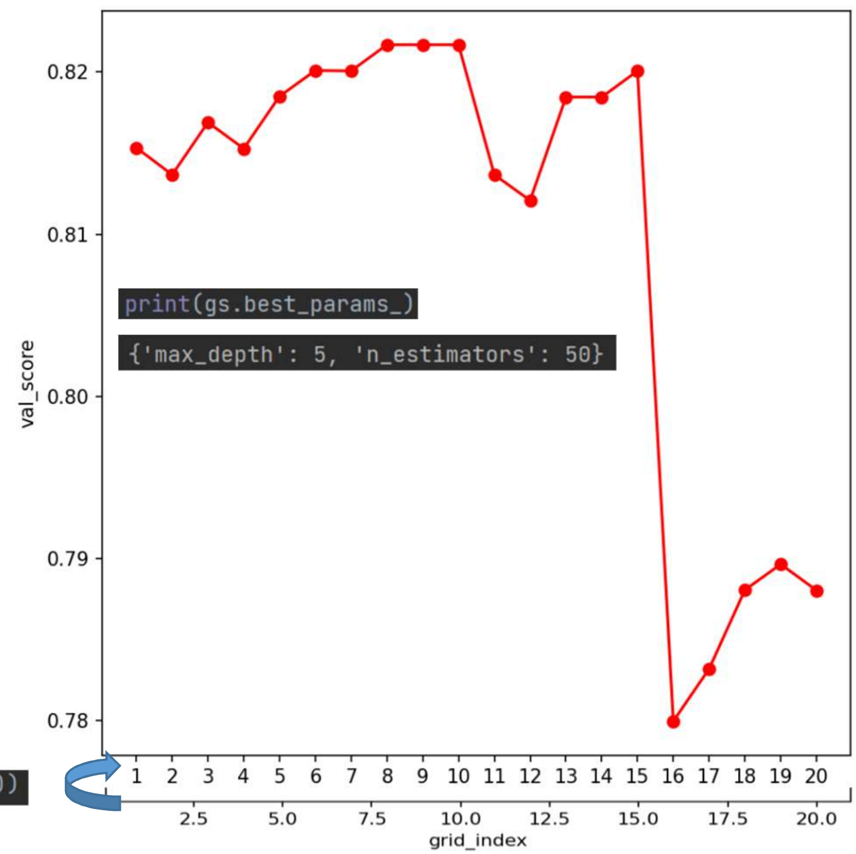
그리드서치(n_estimators, max_depth)

max_depth / n_estimators	3	5	7	10
10	1	6	11	16
30	2	7	12	17
50	3	8	13	18
70	4	9	14	19
100	5	10	15	20

랜덤포레스트 (max_depth, n_estimators)

```
Train-Eval: 0.8346709470304976
Test-Eval : 0.8097014925373134
```

```
plt.xticks(range(1, 21), range(1, 21))
```



엑스트라트리(Extra Trees)

- 의사결정나무에 더 무작위성을 추가 (Extremely Randomized Trees)
- 랜덤포레스트와 비슷하게 일부 특징들을 무작위로 선택하여 의사결정트리 생성
- 무작위로 선택된 특징들마다의 분할 기준도 무작위로 결정
- 학습 데이터 구성 시 부트스트래핑 샘플링이 아닌 전체 데이터 활용
- 특징을 무작위로 선택하기 때문에 랜덤포레스트 보다 연산속도가 빠름

sklearn.ensemble.ExtraTreesClassifier

class sklearn.ensemble.ExtraTreesClassifier(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

An extra-trees classifier.

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

n_estimators : *int, default=100*

The number of trees in the forest.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

`feature_importances_` **: *ndarray of shape (n_features,)***

The impurity-based feature importances.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html#sklearn.ensemble.ExtraTreesClassifier

특징 중요도

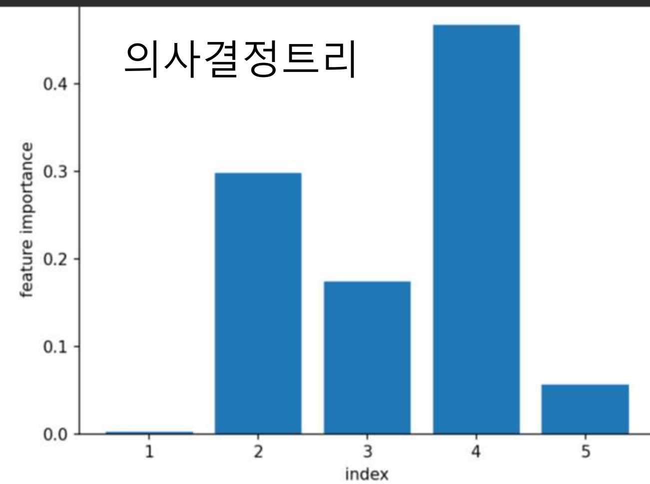
```
import matplotlib.pyplot as plt
```

구현

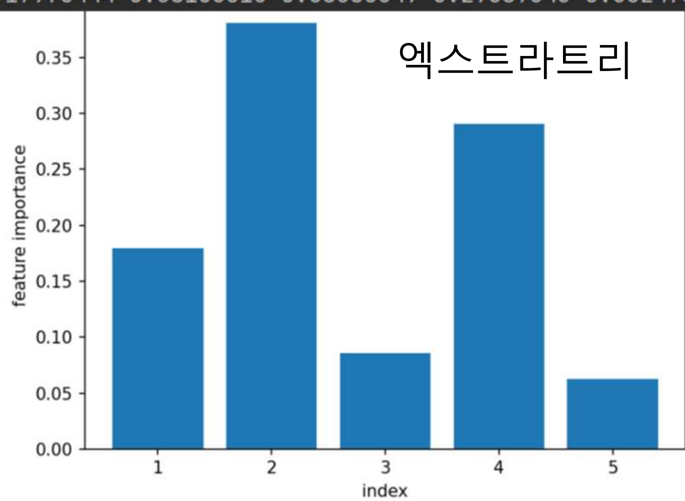
```
plt.show()
```

```
titanic[["sex", "age", "sibsp", "adult_male", "parch"]]
```

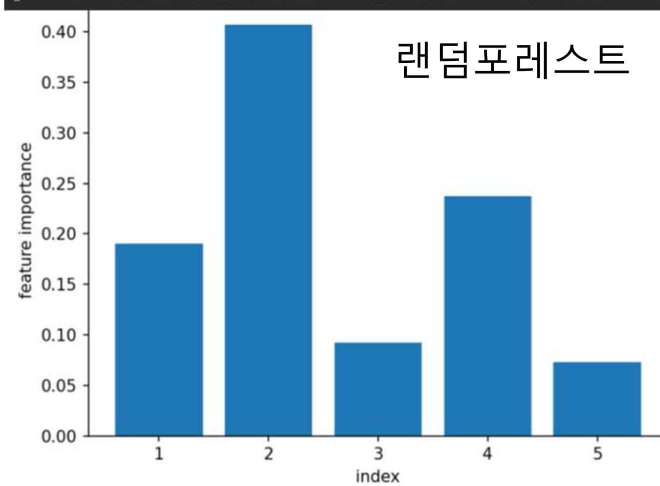
```
[0.00226289 0.29825588 0.17457605 0.46794058 0.05696459]
```




```
[0.17976444 0.38106015 0.08580549 0.29089343 0.06247649]
```




```
[0.19036899 0.4072274  0.09223909 0.23720079 0.07296372]
```



참고자료

- 지능기전공학부 최유경 교수님 자료, https://github.com/sejongresearch/2021.MachineLearning
- 코랩(Colab), https://colab.research.google.com/
- 파이썬(Python), https://www.python.org/doc/
- 사이킷런(sckit-learn), https://scikit-learn.org/stable/index.html
- 판다스(pandas), https://pandas.pydata.org/
- 맷플롯립(matplotlib), https://matplotlib.org/
- 씨본(seaborn), https://seaborn.pydata.org/
- 캐글(Kaggle), https://www.kaggle.com/
- 넘파이(numpy), https://numpy.org/doc/stable/
- 스택오퍼플러우(stackoverflow), https://stackoverflow.com/