

# 타이타닉 생존자 예측

- pclass : 좌석의 등급
- sex : 성별
- age : 나이
- sibsp : 함께 탑승한 형제 및 배우자의 수
- parch : 함께 탑승한 부모 및 자녀의 수
- fare : 요금
- class : 객실 등급
- who : 남자, 여자, 아이
- adult\_male : 성인남자는 True, 그 외 False
- ticket : 티켓 번호
- deck : 선실 번호 첫 알파벳
- embarked : 승선한 항구
- embark\_town : 탑승지 이름
- alive : 생존 여부
- alone : 혼자 탑승 여부(혼자=True, 동승=False)
- survived : 생존 유무(1=생존, 0=사망)

## ✓ 1) 로지스틱 회귀분석 (통계 ver)

```
import seaborn as sns
df = sns.load_dataset('titanic')
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adu
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---

```

```

0  survived      891 non-null    int64
1  pclass        891 non-null    int64
2  sex           891 non-null    object
3  age           714 non-null    float64
4  sibsp         891 non-null    int64
5  parch         891 non-null    int64
6  fare          891 non-null    float64
7  embarked      889 non-null    object
8  class         891 non-null    category
9  who           891 non-null    object
10 adult_male    891 non-null    bool
11 deck          203 non-null    category
12 embark_town   889 non-null    object
13 alive         891 non-null    object
14 alone         891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB

```

# 결측치 전처리

```

df['age'] = df['age'].fillna(df['age'].median())
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])
df['deck'] = df['deck'].fillna(df['deck'].mode()[0])
df['embark_town'] = df['embark_town'].fillna(df['embark_town'].mode()[0])

```

```
df.isnull().sum()
```

```

⇒ survived      0
   pclass        0
   sex           0
   age           0
   sibsp         0
   parch         0
   fare          0
   embarked      0
   class         0
   who           0
   adult_male    0
   deck          0
   embark_town   0
   alive         0
   alone         0
dtype: int64

```

#중복 컬럼 제거

```
df.drop(['pclass','embarked','alive'],axis = 1,inplace = True)
```

# 범주형 변수 더미변수화

```

import pandas as pd
cat_df= df.select_dtypes(include = ['object','category'])
df = pd.get_dummies(df,columns = cat_df.columns)
df

```



	survived	age	sibsp	parch	fare	adult_male	alone	sex_female	sex_male
<b>0</b>	0	22.0	1	0	7.2500	True	False	False	True
<b>1</b>	1	38.0	1	0	71.2833	False	False	True	False
<b>2</b>	1	26.0	0	0	7.9250	False	True	True	False
<b>3</b>	1	35.0	1	0	53.1000	False	False	True	False
<b>4</b>	0	35.0	0	0	8.0500	True	True	False	True
...	...	...	...	...	...	...	...	...	...
<b>886</b>	0	27.0	0	0	13.0000	True	True	False	True
<b>887</b>	1	19.0	0	0	30.0000	False	True	True	False
<b>888</b>	0	28.0	1	2	23.4500	False	False	True	False
<b>889</b>	1	26.0	0	0	30.0000	True	True	False	True
<b>890</b>	0	32.0	0	0	7.7500	True	True	False	True

891 rows × 10 columns

#독립변수, 종속변수 분리

```
df = df.astype(float)
```

```
X = df.drop('survived', axis = 1)
```

```
y = df['survived']
```

# 홀드아웃 기법

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify = y, test_size = 0.3, random_state=42)
```

```
import statsmodels.api as sm
```

```
model = sm.Logit(y_train, X_train)
```

```
result = model.fit()
```



Warning: Maximum number of iterations has been exceeded.

Current function value: 0.411477

Iterations: 35

/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check your Hessian.  
warnings.warn("Maximum Likelihood optimization failed to converge. Check your Hessian.", ConvergenceWarning)

# 계수(Coefficients): 각 독립변수의 영향력과 방향성

```
#
```

```
print(result.summary())
```



## Logit Regression Results

```

=====
Dep. Variable:      survived      No. Observations:      623
Model:              Logit         Df Residuals:           604
Method:              MLE          Df Model:              18
Date:               Sat, 11 May 2024  Pseudo R-squ.:          0.3820
Time:               16:58:08         Log-Likelihood:         -256.35
converged:           False          LL-Null:              -414.80
Covariance Type:    nonrobust       LLR p-value:           1.591e-56
=====

```

```

=====
              coef      std err          z      P>|z|      [0.025      0.975]
-----
age              -0.0164      0.011      -1.467      0.142      -0.038      0.006
sibsp            -0.7039      0.197      -3.576      0.000      -1.090     -0.318
parch           -0.4039      0.189      -2.132      0.033      -0.775     -0.033
fare              0.0061      0.005       1.317      0.188      -0.003      0.015
adult_male       -0.1258         nan         nan         nan         nan         nan
alone           -0.6674      0.364      -1.833      0.067      -1.381      0.046
sex_female        2.5662         nan         nan         nan         nan         nan
sex_male          1.8786         nan         nan         nan         nan         nan
class_First        1.8540      6.47e+06      2.87e-07      1.000     -1.27e+07      1.27e+07
class_Second       1.7602      6.47e+06      2.72e-07      1.000     -1.27e+07      1.27e+07
class_Third        0.8306      6.27e+06      1.32e-07      1.000     -1.23e+07      1.23e+07
who_child          2.6796         nan         nan         nan         nan         nan
who_man           -0.1258         nan         nan         nan         nan         nan
who_woman          1.8910         nan         nan         nan         nan         nan
deck_A            -4.1770         nan         nan         nan         nan         nan
deck_B            -3.6525         nan         nan         nan         nan         nan
deck_C            -5.1112         nan         nan         nan         nan         nan
deck_D            -3.6957         nan         nan         nan         nan         nan
deck_E            -3.3000         nan         nan         nan         nan         nan
deck_F            -4.6859         nan         nan         nan         nan         nan
deck_G            29.0671      3.13e+07      9.29e-07      1.000     -6.13e+07      6.13e+07
embark_town_Chерbourg  1.7019         nan         nan         nan         nan         nan
embark_town_Queenstown  1.3805         nan         nan         nan         nan         nan
embark_town_Southampton  1.3624         nan         nan         nan         nan         nan
=====

```

```

# 오즈비 확인(coef)]
import numpy as np
np.exp(result.params)
# class_First은 오즈비가 6.385118e+00
# class_Third은 오즈비가 2.294805e+00
# 즉 calss_First일 때 calss_Third일때보다 생존확률 3배가 증가함을 알 수 있다.

```



```

age              9.837217e-01
sibsp            4.946554e-01
parch            6.676970e-01
fare             1.006116e+00
adult_male       8.817980e-01
alone            5.130438e-01
sex_female       1.301628e+01
sex_male         6.544185e+00
class_First      6.385118e+00
class_Second     5.813370e+00
class_Third      2.294805e+00
who_child        1.457898e+01
who_man          8.817980e-01

```

```

who_woman          6.625920e+00
deck_A             1.534513e-02
deck_B             2.592551e-02
deck_C             6.028665e-03
deck_D             2.483081e-02
deck_E             3.688138e-02
deck_F             9.224820e-03
deck_G             4.204038e+12
embark_town_Chernbourg 5.484351e+00
embark_town_Queenstown 3.976721e+00
embark_town_Southampton 3.905640e+00
dtype: float64

```

```

y_pred = result.predict(X_test)
y_pred # sm 모델은 확률값을 반환

```

```

↩ 703    0.090823
   329    0.982835
   844    0.101128
   681    0.699416
    9     0.945127
   ...
  412    0.868031
  846    0.000424
   140    0.669092
   667    0.085435
   205    1.000000
Length: 268, dtype: float64

```

```

# 이진 분류로 반환
y_pred = np.around(y_pred)

```

```

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred) #순서 기억!
from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_pred)
print(lr_report)

```

```

↩
              precision    recall  f1-score   support

    0.0         0.85        0.83        0.84         165
    1.0         0.74        0.77        0.75         103

 accuracy                   0.81         268
 macro avg              0.79        0.80        0.80         268
 weighted avg           0.81        0.81        0.81         268

```

## 분류분석 평가

- Accuracy(정확도)

- 의미 : 전체 데이터 중 모델이 올바르게 예측한 비율
  - 식 :  $(TP + TN) / (TP + FP + FN + TN)$
  - Sensitivity(민감도, 재현도)
    - 의미 : 실제 양성일 때 양성이라고 예측한 비율
    - 식 :  $TP / (TP + FN)$
  - Precision(정밀도)
    - 의미 : 모델이 긍정으로 예측한 샘플 중 실제 긍정인 비율
    - 식 :  $TP / (TP + FP)$
  - f1-score
    - 정밀도와 민감도의 조화평균, 정밀도와 민감도 간의 균형을 고려할 때 유용
    - 식 :  $(2 \times (\text{precision} \times \text{sensitivity})) / (\text{precision} + \text{sensitivity})$
- 

- TP = 실제 긍정인 경우 모델이 긍정으로 예측한 수
- TN = 실제 부정인 경우 모델이 부정으로 올바르게 예측한 수
- FP = 실제 부정인데 모델이 긍정으로 잘못 예측한 수
- FN = 실제 긍정인 경우 모델이 부정으로 잘못 예측한 수

## ✓ 로지스틱 회귀분석(ML 라이브러리 ver)

```

import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# 데이터 로드
df = sns.load_dataset('titanic')

# 결측치 전처리
df['age'] = df['age'].fillna(df['age'].median())
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])
df['deck'] = df['deck'].fillna(df['deck'].mode()[0])
df['embark_town'] = df['embark_town'].fillna(df['embark_town'].mode()[0])

# 중복 변수 제거
df.drop(['pclass', 'embarked', 'alive'], axis = 1, inplace = True)

# 라벨인코더로 범주형 변수 수치화

import pandas as pd
from sklearn.preprocessing import LabelEncoder
print('라벨인코딩 처리')
cat_df = df.select_dtypes(include = ['object', 'category'])

encoders= {}
for col in cat_df.columns :
    encoder = LabelEncoder()
    df[col] = encoder.fit_transform(df[col])
    encoders[col] = encoder # 각 컬럼의 encoder 저장

# 독립변수와 종속변수 분리
X = df.drop('survived', axis=1)
y = df['survived']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 모델 생성 및 훈련
model = LogisticRegression(max_iter=400)
model.fit(X_train, y_train)

# 예측
y_pred = model.predict(X_test)

# 성능 평가
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

```



라벨인코딩 처리

Accuracy: 0.8156424581005587

Confusion Matrix:

```
[[91 14]
 [19 55]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.87	0.85	105
1	0.80	0.74	0.77	74
accuracy			0.82	179
macro avg	0.81	0.80	0.81	179
weighted avg	0.81	0.82	0.81	179

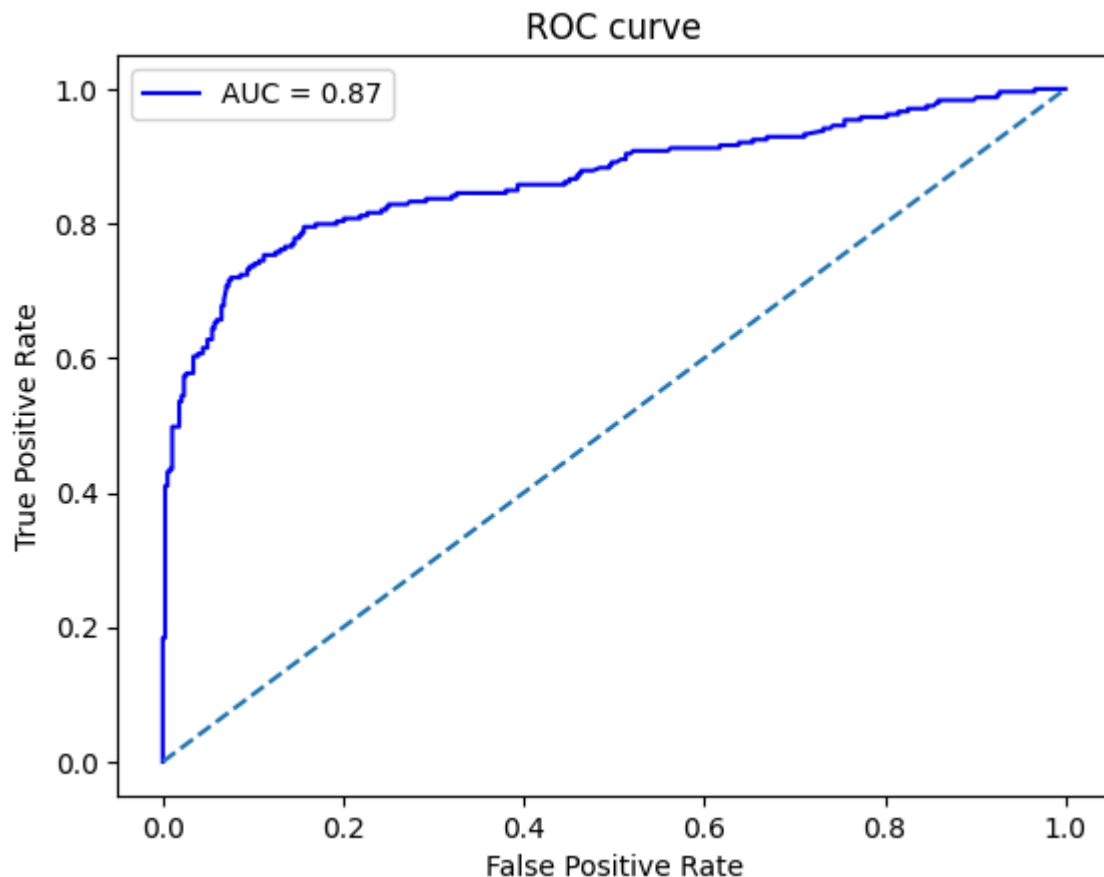
# ROC 그래프 그리기

# ROC 곡선이란 X축 FPR(1-특이도), y축은 재현율을 가지는 그래프로

# TPR = 1, FPR은 0일때 가장 잘 분류된 것을 의미

```
from sklearn.metrics import roc_curve, auc
y_train_pred= result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc= auc(fpr, tpr)
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

↗ <matplotlib.legend.Legend at 0x796ba0786260>





## ✓ k-fold 교차검증

- 모델의 성능을 평가하고 과적합을 방지하기 위해 데이터를 여러 번 나누어 학습 및 평가하는 방법
- 데이터셋을 K개의 폴드(fold)로 나누고, 각 폴드를 한 번씩 검증 데이터로 사용하며, 나머지 폴드를 학습 데이터로 사용하여 모델을 K번 학습 및 평가

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

# 데이터 로드
df = sns.load_dataset('titanic')

# 결측치 전처리
df['age'] = df['age'].fillna(df['age'].median())
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])
df['deck'] = df['deck'].fillna(df['deck'].mode()[0])
df['embark_town'] = df['embark_town'].fillna(df['embark_town'].mode()[0])

# 중복 변수 제거
df.drop(['pclass', 'embarked', 'alive'], axis=1, inplace=True)

# 라벨인코더로 범주형 변수 수치화
cat_df = df.select_dtypes(include=['object', 'category'])
for col in cat_df.columns:
    encoder = LabelEncoder()
    df[col] = encoder.fit_transform(df[col])

# 독립변수와 종속변수 분리
X = df.drop('survived', axis=1)
y = df['survived']

# K-fold 교차 검증 설정
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 결과를 저장할 리스트 초기화
best_accuracy = 0
best_report = None

# K-fold 교차 검증 수행
for train_index, test_index in kf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # 모델 생성 및 훈련
    model = LogisticRegression(max_iter=400)
    model.fit(X_train, y_train)

    # 예측 및 성능 평가
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # 최적 모델 업데이트
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_report = classification_report(y_test, y_pred, output_dict=True)

# 최적 모델의 리포트 출력
```

```
print("Best Classification Report:\n", pd.DataFrame(best_report))
print("Best Accuracy:", best_accuracy)
```

```
⇒ Classification Report for fold 1:
              0          1  accuracy  macro avg  weighted avg
precision    0.854545  0.768116  0.821229    0.811331    0.821229
recall       0.854545  0.768116  0.821229    0.811331    0.821229
f1-score     0.854545  0.768116  0.821229    0.811331    0.821229
support     110.000000  69.000000  0.821229  179.000000    179.000000
Classification Report for fold 2:
              0          1  accuracy  macro avg  weighted avg
precision    0.831858  0.753846  0.803371    0.792852    0.802056
recall       0.854545  0.720588  0.803371    0.787567    0.803371
f1-score     0.843049  0.736842  0.803371    0.789946    0.802476
support     110.000000  68.000000  0.803371  178.000000    178.000000
Classification Report for fold 3:
              0          1  accuracy  macro avg  weighted avg
precision    0.825000  0.810345  0.820225    0.817672    0.819401
recall       0.900000  0.691176  0.820225    0.795588    0.820225
f1-score     0.860870  0.746032  0.820225    0.803451    0.816999
support     110.000000  68.000000  0.820225  178.000000    178.000000
Classification Report for fold 4:
              0          1  accuracy  macro avg  weighted avg
precision    0.843478  0.793651  0.825843    0.818565    0.824443
recall       0.881818  0.735294  0.825843    0.808556    0.825843
f1-score     0.862222  0.763359  0.825843    0.812791    0.824454
support     110.000000  68.000000  0.825843  178.000000    178.000000
Classification Report for fold 5:
              0          1  accuracy  macro avg  weighted avg
precision    0.854545  0.779412  0.825843    0.816979    0.825421
recall       0.862385  0.768116  0.825843    0.815251    0.825843
f1-score     0.858447  0.773723  0.825843    0.816085    0.825605
support     109.000000  69.000000  0.825843  178.000000    178.000000
Accuracies: [0.8212290502793296, 0.8033707865168539, 0.8202247191011236, 0.8258426966292135,
Mean Accuracy: 0.8193019898311468
```

## ✓ KNN(K-Near-Neighbors)

- 새로운 관측값이 주어질 때 기존 데이터에서 가장 속성이 비슷한 k개의 이웃을 찾음
- 가까운 이웃들이 가진 목표 값과 같은 값으로 분류
- 즉, K값에 따라 예측의 정확도가 달라지므로 적절한 K 값을 찾아야 함

```
import seaborn as sns
iris = sns.load_dataset('iris')
print(iris.head())

sns.set()
sns.pairplot(iris, hue = 'species', size = 2.5)
```



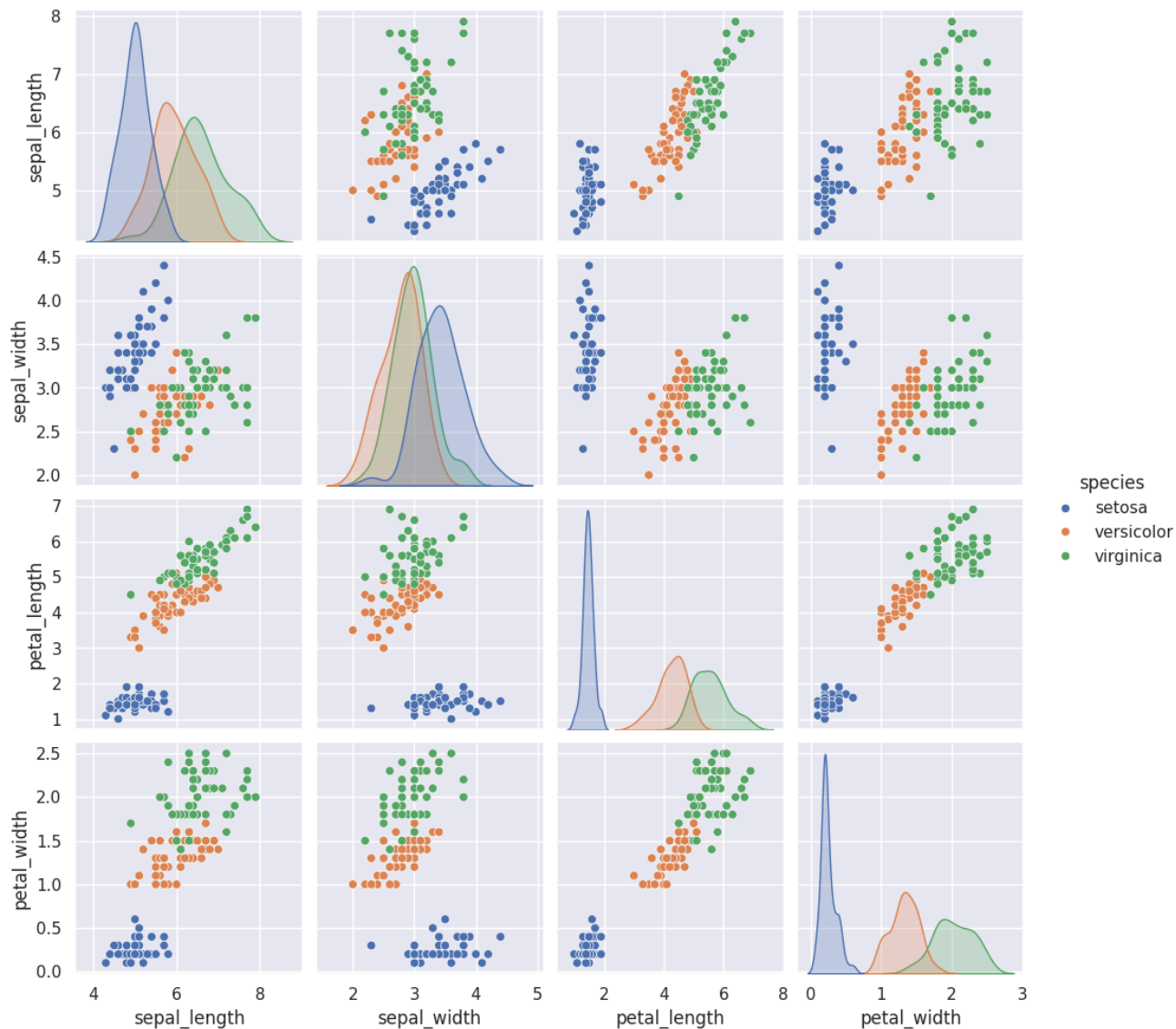
```

sepal_length sepal_width petal_length petal_width species
0          5.1         3.5         1.4         0.2  setosa
1          4.9         3.0         1.4         0.2  setosa
2          4.7         3.2         1.3         0.2  setosa
3          4.6         3.1         1.5         0.2  setosa
4          5.0         3.6         1.4         0.2  setosa

```

/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:2100: UserWarning: The `size` parameter is deprecated. Use `figure` or `ax` instead.  
warnings.warn(msg, UserWarning)


<seaborn.axisgrid.PairGrid at 0x7f6f04184ee0>



```
X=iris.drop('species', axis=1)
y=iris['species']
```

```
from sklearn.preprocessing import LabelEncoder
import numpy as np
encoder = LabelEncoder()
y= encoder.fit_transform(iris['species'].values)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```



	sepal_length	sepal_width	petal_length	petal_width
33	5.5	4.2	1.4	0.2
20	5.4	3.4	1.7	0.2
115	6.4	3.2	5.3	2.3
124	6.7	3.3	5.7	2.1
35	5.0	3.2	1.2	0.2
..	...	...	...	...
41	4.5	2.3	1.3	0.3
92	5.8	2.6	4.0	1.2
26	5.0	3.4	1.6	0.4
3	4.6	3.1	1.5	0.2
42	4.4	3.2	1.3	0.2

[105 rows x 4 columns]

	sepal_length	sepal_width	petal_length	petal_width
148	6.2	3.4	5.4	2.3
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
106	4.9	2.5	4.5	1.7
75	6.6	3.0	4.4	1.4
71	6.1	2.8	4.0	1.3
108	6.7	2.5	5.8	1.8
68	6.2	2.2	4.5	1.5
111	6.4	2.7	5.3	1.9
16	5.4	3.9	1.3	0.4
15	5.7	4.4	1.5	0.4
127	6.1	3.0	4.9	1.8
8	4.4	2.9	1.4	0.2
57	4.9	2.4	3.3	1.0
47	4.6	3.2	1.4	0.2
76	6.8	2.8	4.8	1.4
141	6.9	3.1	5.1	2.3
66	5.6	3.0	4.5	1.5
74	6.4	2.9	4.3	1.3

147	6.5	3.0	5.2	2.0
102	7.1	3.0	5.9	2.1
12	4.8	3.0	1.4	0.1
65	6.7	3.1	4.4	1.4
121	5.6	2.8	4.9	2.0
94	5.6	2.7	4.2	1.3
56	6.3	3.3	4.7	1.6
98	5.1	2.5	3.0	1.1
144	6.7	3.3	5.7	2.5
43	5.0	3.5	1.6	0.6
113	5.7	2.5	5.0	2.0
11	4.8	3.4	1.6	0.2
0	5.1	3.5	1.4	0.2
62	6.0	2.2	4.0	1.0
53	5.5	2.3	4.0	1.3
107	7.3	2.9	6.3	1.8
112	6.8	3.0	5.5	2.1
37	4.9	3.6	1.4	0.1
25	5.0	3.0	1.6	0.2
1	4.9	3.0	1.4	0.2
50	7.0	3.2	4.7	1.4
120	6.9	3.2	5.7	2.3
149	5.9	3.0	5.1	1.8
54	6.5	2.8	4.6	1.5

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
knn_report = metrics.classification_report(y_test, y_pred)
print(knn_report)
```

```
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	15
1	0.94	1.00	0.97	15
2	1.00	0.93	0.97	15
accuracy			0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

## ✓ SVM(Support Vector Machine)

- 분류 및 회귀 모두 사용 가능
- 데이터를 분류하기 위해 클래스 사이에 최대 마진을 가진 경계를 찾음

옵션:

- **kernel**: 선형('linear')/비선형('rbf')
  - 데이터가 선형적으로 구분 가능한 경우 linear, 선형적으로 구분 불가능한 경우 rbf 사용
- **C**: 선형 SVM 초모수값
  - 규제 파라미터로, 모델의 복잡도와 데이터 포인트의 오차 허용 정도를 조절. 값이 클수록 모델이 데이터를 더 정확하게 맞추나 과적합 위험이 커짐
- **gamma**
  - rbf 커널의 하이퍼파라미터, 결정 경계를 얼마나 구체적으로 만들지 결정
- **random\_state**: 초기난수 (모델의 재현성 보장)

```

from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report

# 데이터 로드
iris = load_iris()
X = iris.data
sc = StandardScaler()
X = sc.fit_transform(X)
y = iris.target

# 훈련 데이터와 테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# SVM 모델 생성 및 훈련
svm = SVC(kernel='linear') # 선형 커널 사용

svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

# 모델 평가
print("Accuracy:", svm.score(X_test, y_test))
svm_report = classification_report(y_test, y_pred)
print(svm_report)

```



Accuracy: 0.9777777777777777

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	0.92	0.96	13
2	0.93	1.00	0.96	13
accuracy			0.98	45
macro avg	0.98	0.97	0.97	45
weighted avg	0.98	0.98	0.98	45

## ✓ 최적의 파라미터 찾기

- GridSearch
- 하이퍼파라미터 값들의 조합을 모두 시도하여 최적의 성능을 보이는 하이퍼파라미터 값을 찾는 방식

```

from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report
from sklearn import metrics

# 데이터 로드
iris = load_iris()
X = iris.data
y = iris.target

# 훈련 데이터와 테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# SVM 모델을 위한 그리드 서치 매개변수 설정
param_grid = {
    'C': [0.1, 1, 10, 100], # C 매개변수
    'gamma': [1, 0.1, 0.01, 0.001], # 감마 매개변수
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'] # 커널
}


# GridSearchCV 객체 생성
grid_search = GridSearchCV(SVC(), param_grid, refit=True)

# 그리드 서치를 통한 모델 학습
grid_search.fit(X_train, y_train)

# 최적의 모델과 매개변수
print("Best parameters:", grid_search.best_params_)
print("Best estimator:", grid_search.best_estimator_)

# 테스트 데이터셋을 사용하여 모델 평가
y_pred = grid_search.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
svm_report = classification_report(y_test, y_pred)
print(svm_report)

```

 Best parameters: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}  
 Best estimator: SVC(C=100, gamma=0.01)  
 Accuracy: 1.0

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13



accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

## ✓ 의사결정나무 (Decision Tree)

### 특징

- 분리규칙을 통한 예측 및 분류
  - 연속형인 경우 분리 기준보다 작으면 왼쪽, 크면 오른쪽
  - 범주형인 경우 전체 범주를 두 개의 부분집합으로 분리
- 순수도와 불순도
  - 목표변수를 얼마나 잘 구분하는가에 따라 불순도 및 순수도를 측정
- 불순도 척도 종류
  - 범주형 : 카이제곱, 지니지수, 엔트로피 지수
  - 연속형 : 분산분석에 의한 F통계량, 분산의 감소량

### 장점

- If-then으로 표현되는 규칙 -> 이해가 쉽고 해석이 쉬움
- 데이터 스케일에 영향 받지 않고 설명력이 좋음
- 비모수적 모형으로 등분산, 정규성 등의 가정이 필요하지 않음

### 단점

- 비연속성 : 경계점 근방에서 오류 가능성이 존재

```
# tips 데이터를 로드하여 sex를 목적변수로 하여 분류하는 모델을 의사결정 나무로 작성
import seaborn as sns
import pandas as pd
tips = sns.load_dataset('tips')
tips.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  -
0   total_bill  244 non-null    float64
1   tip         244 non-null    float64
2   sex        244 non-null    category
3   smoker     244 non-null    category
4   day        244 non-null    category
```

```

5   time          244 non-null    category
6   size          244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB

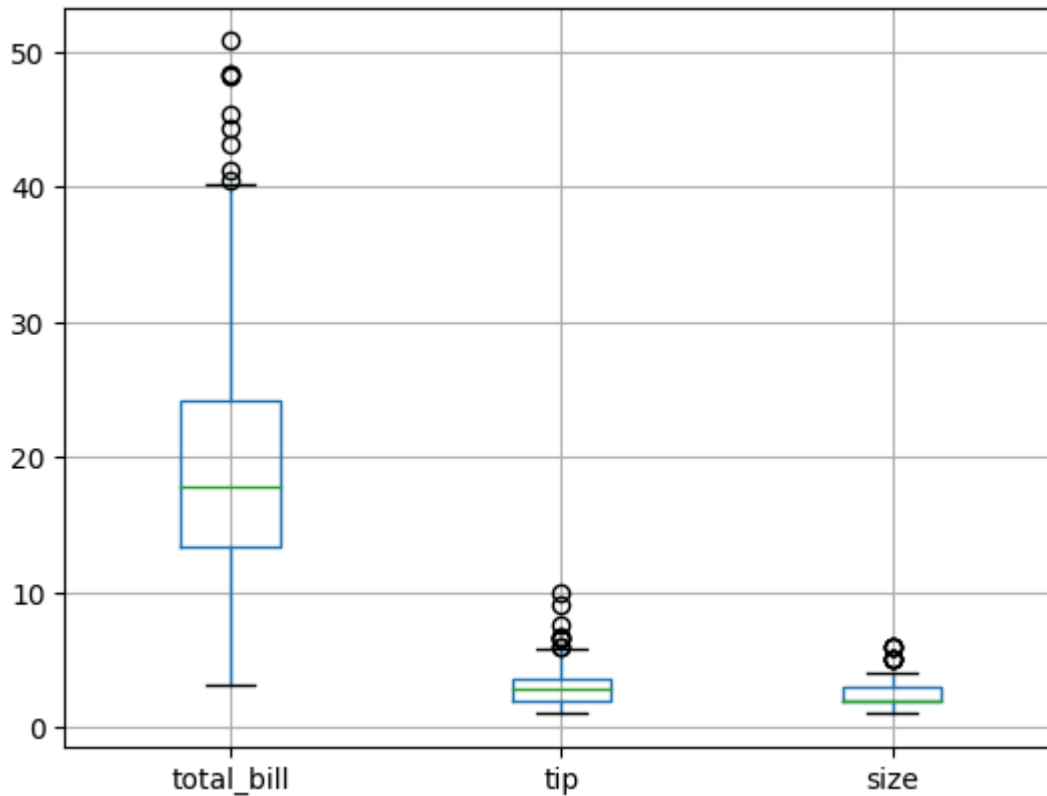
```

코딩을 시작하거나 AI로 코드를 생성하세요.

```
tips.boxplot()
```



<Axes: >



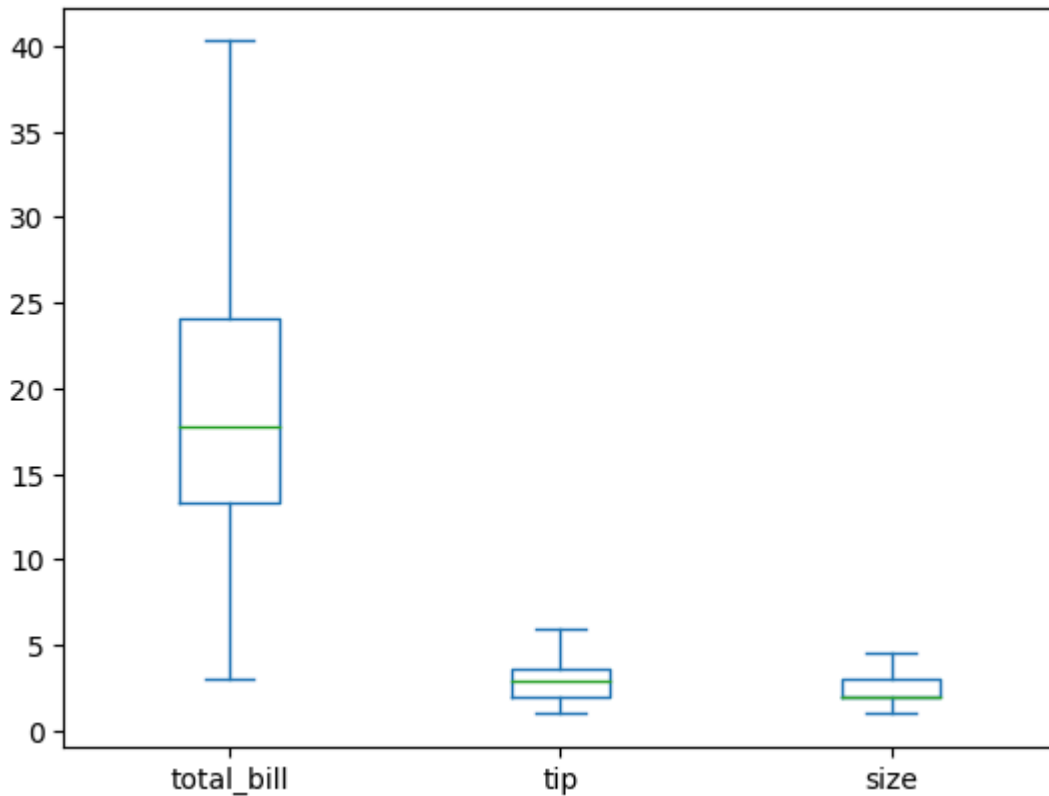
```

def remove_outlier(df, columns):
    for i in columns:
        q1 = df[i].quantile(0.25)
        q3 = df[i].quantile(0.75)
        IQR = q3 - q1
        lower_bound = q1 - IQR*1.5
        upper_bound = q3 + IQR*1.5
        df[i] = df[i].clip(lower_bound, upper_bound)
    return df

numeric_col = tips.select_dtypes(exclude = ['object', 'category', 'bool']).columns
data = remove_outlier(tips, numeric_col)
data.plot(kind = 'box')

```

↩ <Axes: >



```
X = tips.drop('sex',axis =1)
y = tips['sex']
```

```
X_dummies = pd.get_dummies(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_dummies,y,test_size = 0.3, random_state = 1)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
scaled_train = sc.fit_transform(X_train)
scaled_test = sc.transform(X_test)
#scaled_df = pd.DataFrame(scaled, columns = X.columns)
```

```
print(scaled_train)
```

```
↩ [[ 3.27957576  2.93016025  1.4480456  ... -0.68228824 -0.6638358
    0.6638358 ]
 [ 2.82704615 -0.34414084  0.43500958 ... -0.68228824 -0.6638358
    0.6638358 ]
 [-0.86956223 -1.01757865 -0.57802643 ... -0.68228824  1.50639662
   -1.50639662]
 ...
 [-0.23123331  0.42992561 -0.57802643 ... -0.68228824  1.50639662
   -1.50639662]
 [-1.07473939 -1.3117239  -0.57802643 ... -0.68228824 -0.6638358
    0.6638358 ]
 [-0.29278646  0.09707704  0.43500958 ... -0.68228824 -0.6638358
    0.6638358 ]]
```