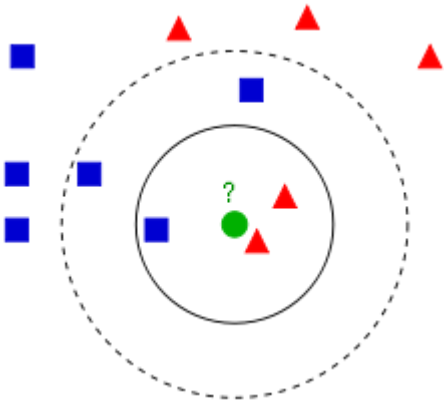
최근접 이웃(K-Nearest Neighbor)

- 특별한 예측 모델 없이 가장 가까운 데이터 포인트를 기반으로 예측을 수행하는 방법
- 분류와 회귀 모두 지원




```
import pandas as pd
import numpy as np
import multiprocessing
import matplotlib.pyplot as plt
plt.style.use(['seaborn-whitegrid'])
```

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.manifold import TSNE
from sklearn.datasets import load_iris, load_breast_cancer, load_wine
from sklearn.datasets import load_boston, fetch_california_housing
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline, Pipeline
```

K 최근접 이웃 분류

- 입력 데이터 포인트와 가장 가까운 k개의 훈련 데이터 포인트가 출력
- k개의 데이터 포인트 중 가장 많은 클래스가 예측 결과

붓꽃 데이터

```
iris = load_iris()
```

```
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['Target'] = iris.target
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2

```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform(X_test)
```

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.9583333333333334
평가 데이터 점수: 1.0
```

```
model = KNeighborsClassifier()
model.fit(X_train_scale, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.30833333333333335
평가 데이터 점수: 0.43333333333333335
```

```
cross_validate(
    estimator=KNeighborsClassifier(),
    X=X, y=y,
    cv=5,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   5 out of   5 | elapsed:    1.0s finished
{'fit_time': array([0.0015676 , 0.00156307, 0.00086594, 0.00075507, 0.00075579]),
 'score_time': array([0.00344396, 0.00344849, 0.00292969, 0.00335789, 0.00197649]),
 'test_score': array([0.96666667, 1.        , 0.93333333, 0.96666667, 1.        ])}
```

```
param_grid = [{'n_neighbors': [3, 5, 7],
               'weights': ['uniform', 'distance'],
               'algorithm': ['ball_tree', 'kd_tree', 'brute']}]
```

```
gs = GridSearchCV(
    estimator=KNeighborsClassifier(),
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
gs.fit(X, y)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  90 out of  90 | elapsed:    0.3s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=2,
             param_grid=[{'algorithm': ['ball_tree', 'kd_tree', 'brute'],
                          'n_neighbors': [3, 5, 7],
                          'weights': ['uniform', 'distance']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

```
gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='ball_tree', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

```
print('GridSearchCV best score: {}'.format(gs.best_score_))
```

```
GridSearchCV best score: 0.9800000000000001
```

```
def make_meshgrid(x, y, h=.02):
  x_min, x_max = x.min()-1, x.max()+1
  y_min, y_max = y.min()-1, y.max()+1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                       np.arange(y_min, y_max, h))

  return xx, yy

def plot_contours(clf, xx, yy, **params):
  Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)
  out = plt.contourf(xx,yy, Z, **params)

  return out
```

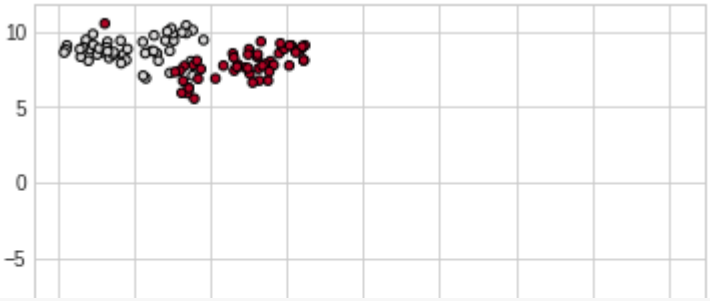
```
tsne = TSNE(n_components=2)
X_comp = tsne.fit_transform(X)
```

```
iris_comp_df = pd.DataFrame(data=X_comp)
iris_comp_df['Target'] = y
iris_comp_df
```

	0	1	Target
0	16.188223	-14.229498	0
1	18.800316	-13.376467	0
2	18.656727	-14.551187	0
3	19.100636	-14.174066	0
4	16.227226	-14.648671	0
...	...	...	...
145	-7.505892	8.471780	2
146	-10.598353	7.509082	2
147	-8.443097	8.263677	2
148	-7.210024	6.611504	2
149	-11.763679	6.723948	2

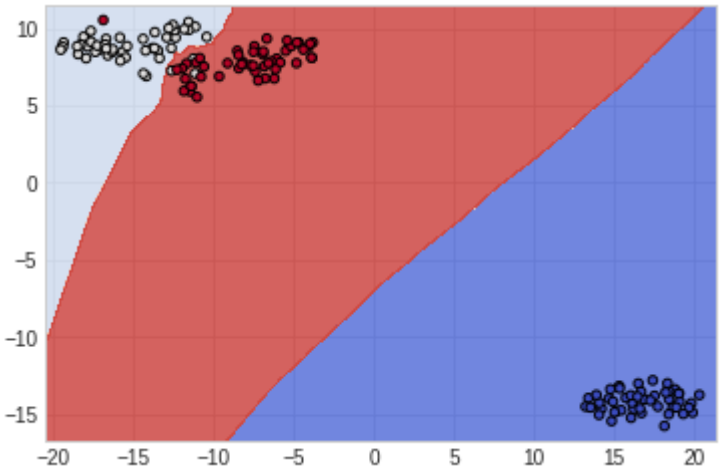
150 rows × 3 columns

```
plt.scatter(X_comp[:, 0], X_comp[:, 1],
            c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = KNeighborsClassifier()
model.fit(X_comp, y)
predict = model.predict(X_comp)
```

```
xx, yy = make_meshgrid(X_comp[:, 0], X_comp[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_comp[:, 0], X_comp[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



유방암 데이터

```
cancer = load_breast_cancer()
```

```
cancer_df = pd.DataFrame(data=cancer.data, columns=cancer.feature_names)
cancer_df['target'] = cancer.target
cancer_df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430

```
X, y = cancer.data, cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791

```
cancer_train_df = pd.DataFrame(data=X_train, columns=cancer.feature_names)
cancer_train_df['target'] = y_train
cancer_train_df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	10.750	14.97	68.26	355.3	0.07793	0.05139	0.02251	0.007875
1	23.210	26.97	153.50	1670.0	0.09509	0.16820	0.19500	0.123700
2	9.042	18.90	60.07	244.5	0.09968	0.19720	0.19750	0.049080
3	13.660	19.13	89.46	575.3	0.09057	0.11470	0.09657	0.048120
4	14.970	16.95	96.22	685.9	0.09855	0.07885	0.02602	0.037810
...	...	...	...	...	...	...	...	...
450	11.410	10.82	73.34	403.3	0.09373	0.06685	0.03512	0.026230
451	27.420	26.27	186.90	2501.0	0.10840	0.19880	0.36350	0.168900
452	19.730	19.82	130.70	1206.0	0.10620	0.18490	0.24170	0.097400
453	11.270	15.50	73.38	392.0	0.08365	0.11140	0.10070	0.027570
454	18.660	17.12	121.40	1077.0	0.10540	0.11000	0.14570	0.086650

455 rows × 31 columns

```
cancer_test_df = pd.DataFrame(data=X_test, columns=cancer.feature_names)
cancer_test_df['target'] = y_test
cancer_test_df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900
1	13.850	15.18	88.99	587.4	0.09516	0.07688	0.044790	0.037110
2	10.900	12.96	68.69	366.8	0.07515	0.03718	0.003090	0.006588
3	21.090	26.57	142.70	1311.0	0.11410	0.28320	0.248700	0.149600
4	13.200	15.82	84.07	537.3	0.08511	0.05251	0.001461	0.003261
...	...	...	...	...	...	...	...	...
109	19.210	18.57	125.50	1152.0	0.10530	0.12670	0.132300	0.089940
110	11.600	12.84	74.34	412.6	0.08983	0.07525	0.041960	0.033500
111	11.500	18.45	73.28	407.4	0.09345	0.05991	0.026380	0.020690
112	8.734	16.84	55.27	234.3	0.10390	0.07428	0.000000	0.000000

```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform(X_test)
```

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.9384615384615385
평가 데이터 점수: 0.9298245614035088
```

```
model = KNeighborsClassifier()
model.fit(X_train_scale, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train_scale, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test_scale, y_test)))
```

```
학습 데이터 점수: 0.9824175824175824
평가 데이터 점수: 0.9473684210526315
```

```
estimator = make_pipeline(
    StandardScaler(),
```

```
    KNeighborsClassifier()
)
```

```
cross_validate(
    estimator=estimator,
    X=X, y=y,
    cv=5,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   5 out of   5 | elapsed:    1.0s finished
{'fit_time': array([0.00469589, 0.00460935, 0.0032146 , 0.00355482, 0.00311255]),
 'score_time': array([0.01261187, 0.01069021, 0.01152253, 0.01135063, 0.01077199]),
 'test_score': array([0.96491228, 0.95614035, 0.98245614, 0.95614035, 0.96460177])}
```

```
pipe = Pipeline(
    [('scaler', StandardScaler()),
     ('model', KNeighborsClassifier())]
)
```

```
param_grid = [{'model__n_neighbors': [3, 5, 7],
               'model__weights': ['uniform', 'distance'],
               'model__algorithm': ['ball_tree', 'kd_tree', 'brute']}]
```

```
gs = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
gs.fit(X, y)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  90 out of  90 | elapsed:    0.7s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('model',
                                        KNeighborsClassifier(algorithm='auto',
                                                             leaf_size=30,
                                                             metric='minkowski',
                                                             metric_params=None,
                                                             n_jobs=None,
                                                             n_neighbors=5, p=2,
                                                             weights='uniform'))],
                                verbose=False),
             iid='deprecated', n_jobs=2,
```

```
                param_grid=[{'model__algorithm': ['ball_tree', 'kd_tree', 'brute'],
                             'model__n_neighbors': [3, 5, 7],
                             'model__weights': ['uniform', 'distance']}],
                pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                scoring=None, verbose=True)
```

```
gs.best_estimator_
```

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('model',
                 KNeighborsClassifier(algorithm='ball_tree', leaf_size=30,
                                      metric='minkowski', metric_params=None,
                                      n_jobs=None, n_neighbors=7, p=2,
                                      weights='uniform'))],
         verbose=False)
```

```
print('GridSearchCV best score: {}'.format(gs.best_score_))
```

```
GridSearchCV best score: 0.9701288619779538
```

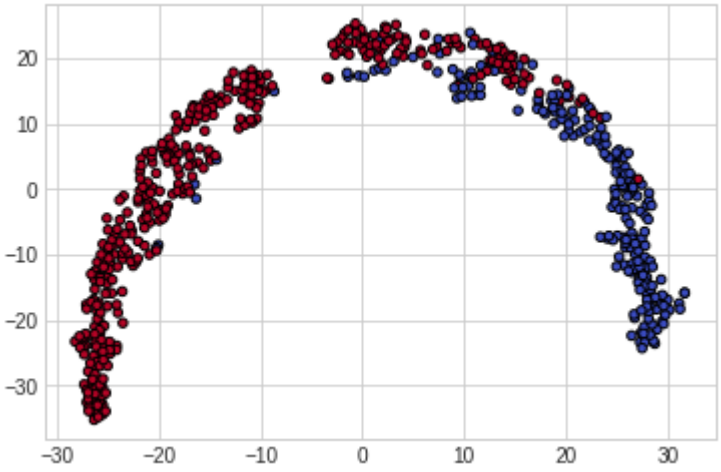
```
tsne = TSNE(n_components=2)
X_comp = tsne.fit_transform(X)
```

```
cancer_comp_df = pd.DataFrame(data=X_comp)
cancer_comp_df['target'] = y
cancer_comp_df
```

	0	1	target
0	31.163456	-17.448296	0
1	28.311378	-17.007872	0
2	27.959192	-12.035782	0
3	-16.328327	-1.452164	0
4	25.752319	-11.579320	0
...	...	...	...
564	27.449287	-18.507101	0
565	27.604269	-13.154164	0
566	23.871012	8.029447	0
567	28.500597	-14.739604	0
568	-25.221497	-33.889301	1

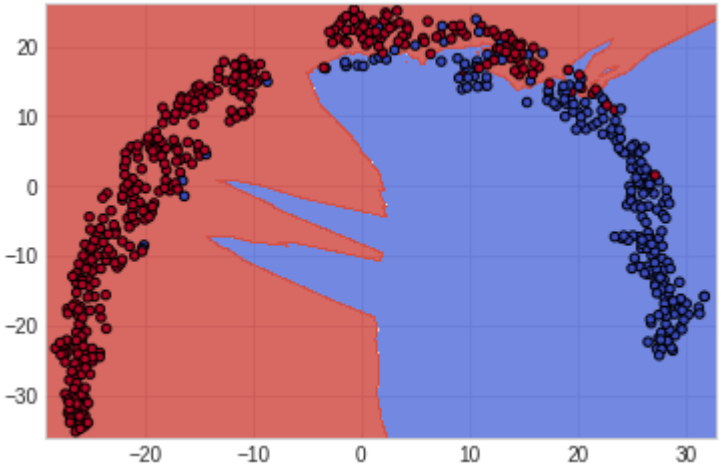
569 rows × 3 columns

```
plt.scatter(X_comp[:, 0], X_comp[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = KNeighborsClassifier()
model.fit(X_comp, y)
predict = model.predict(X_comp)
```

```
xx, yy = make_meshgrid(X_comp[:, 0], X_comp[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_comp[:, 0], X_comp[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



와인 데이터

```
wine = load_wine()
```

```
wine_df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
wine_df['target'] = wine.target
wine_df
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoi
0	14.23	1.71	2.43	15.6	127.0	2.80	3.
1	13.20	1.78	2.14	11.2	100.0	2.65	2.
2	13.16	2.36	2.67	18.6	101.0	2.80	3.
3	14.37	1.95	2.50	16.8	113.0	3.85	3.
4	13.24	2.59	2.87	21.0	118.0	2.80	2.
...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.
174	13.40	3.91	2.48	23.0	102.0	1.80	0.
175	13.27	4.28	2.26	20.0	120.0	1.59	0

```
X, y = wine.data, wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

177	14.13	4.10	2.74	24.5	96.0	2.05	0.

```
wine_train_df = pd.DataFrame(data=X_train, columns=wine.feature_names)
wine_train_df['target'] = y_train
wine_train_df
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoi
0	13.56	1.71	2.31	16.2	117.0	3.15	3.
1	12.22	1.29	1.94	19.0	92.0	2.36	2.
2	12.86	1.35	2.32	18.0	122.0	1.51	1.
3	13.90	1.68	2.12	16.0	101.0	3.10	3.
4	13.16	3.57	2.15	21.0	102.0	1.50	0.
...	...	...	...	...	...	...	
137	12.34	2.45	2.46	21.0	98.0	2.56	2
138	12.70	3.55	2.36	21.5	106.0	1.70	1.
139	12.08	1.13	2.51	24.0	78.0	2.00	1.
140	14.06	2.15	2.61	17.6	121.0	2.60	2.
141	13.24	3.98	2.29	17.5	103.0	2.64	2.

142 rows × 14 columns

```
wine_test_df = pd.DataFrame(data=X_test, columns=wine.feature_names)
wine_test_df['target'] = y_test
wine_test_df
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoid
0	13.28	1.64	2.84	15.5	110.0	2.60	2.6
1	13.05	1.65	2.55	18.0	98.0	2.45	2.4
2	12.51	1.24	2.25	17.5	85.0	2.00	0.5
3	12.72	1.75	2.28	22.5	84.0	1.38	1.7
4	12.37	1.13	2.16	19.0	87.0	3.50	3.1
5	12.37	1.17	1.92	19.6	78.0	2.11	2.0
6	13.86	1.35	2.27	16.0	98.0	2.98	3.1
7	14.38	3.59	2.28	16.0	102.0	3.25	3.1
8	13.83	1.65	2.60	17.2	94.0	2.45	2.9
9	12.00	0.92	2.00	19.0	86.0	2.42	2.2
10	12.79	2.67	2.48	22.0	112.0	1.48	1.3
11	13.17	5.19	2.32	22.0	93.0	1.74	0.6
12	13.73	1.50	2.70	22.5	101.0	3.00	3.2
13	12.58	1.29	2.10	20.0	103.0	1.48	0.5
14	13.41	3.84	2.12	18.8	90.0	2.45	2.6
15	13.11	1.01	1.70	15.0	78.0	2.98	3.1
16	13.36	2.56	2.35	20.0	89.0	1.40	0.5
17	13.88	1.89	2.59	15.0	101.0	3.25	3.5
18	12.08	1.33	2.30	23.6	70.0	2.20	1.5
19	13.94	1.73	2.27	17.4	108.0	2.88	3.5
20	12.29	1.41	1.98	16.0	85.0	2.55	2.5
21	12.67	0.98	2.24	18.0	99.0	2.20	1.9
22	12.87	4.61	2.48	21.5	86.0	1.70	0.6
23	12.45	3.03	2.64	27.0	97.0	1.90	0.5
24	11.82	1.72	1.88	19.5	86.0	2.50	1.6
25	11.79	2.13	2.78	28.5	92.0	2.13	2.2
26	13.73	4.36	2.26	22.5	88.0	1.28	0.4
27	12.93	3.80	2.65	18.6	102.0	2.41	2.4
28	11.84	2.89	2.23	18.0	112.0	1.72	1.3

```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform(X_test)
```

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7746478873239436
평가 데이터 점수: 0.75
```

```
model = KNeighborsClassifier()
model.fit(X_train_scale, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train_scale, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test_scale, y_test)))
```

```
학습 데이터 점수: 0.9859154929577465
평가 데이터 점수: 1.0
```

```
estimator = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier()
)
```

```
cross_validate(
    estimator=estimator,
    X=X, y=y,
    cv=5,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   5 out of   5 | elapsed:    1.0s finished
{'fit_time': array([0.0032866 , 0.00390983, 0.00230169, 0.00187802, 0.00197983]),
 'score_time': array([0.00443602, 0.00481462, 0.00388694, 0.00399375, 0.00345325]),
 'test_score': array([0.94444444, 0.94444444, 0.97222222, 1.        , 0.88571429])}
```

```
pipe = Pipeline(
    [('scaler', StandardScaler()),
     ('model', KNeighborsClassifier())]
)
```

```
param_grid = [{'model__n_neighbors': [3, 5, 7],
               'model__weights': ['uniform', 'distance'],
```

```
              'model__algorithm': ['ball_tree', 'kd_tree', 'brute']}]
```

```
gs = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
gs.fit(X, y)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  90 out of  90 | elapsed:    0.3s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('model',
                                        KNeighborsClassifier(algorithm='auto',
                                                             leaf_size=30,
                                                             metric='minkowski',
                                                             metric_params=None,
                                                             n_jobs=None,
                                                             n_neighbors=5, p=2,
                                                             weights='uniform'))],
                                verbose=False),
             iid='deprecated', n_jobs=2,
             param_grid=[{'model__algorithm': ['ball_tree', 'kd_tree', 'brute'],
                          'model__n_neighbors': [3, 5, 7],
                          'model__weights': ['uniform', 'distance']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

```
gs.best_estimator_
```

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('model',
                 KNeighborsClassifier(algorithm='ball_tree', leaf_size=30,
                                      metric='minkowski', metric_params=None,
                                      n_jobs=None, n_neighbors=7, p=2,
                                      weights='uniform'))],
         verbose=False)
```

```
print('GridSearchCV best score: {}'.format(gs.best_score_))
```

```
GridSearchCV best score: 0.9665079365079364
```

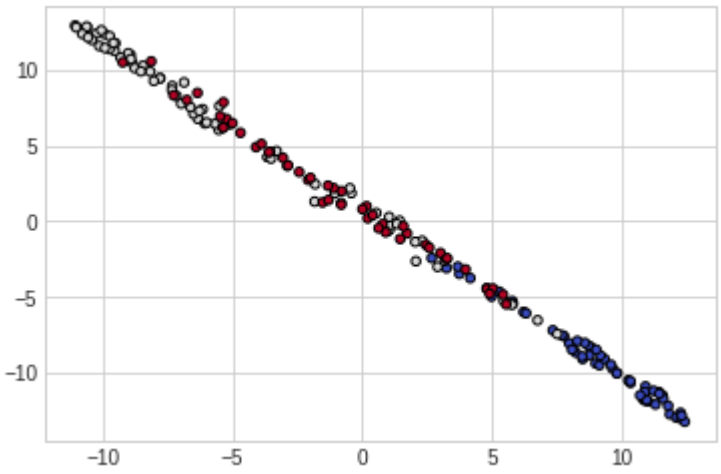
```
tsne = TSNE(n_components=2)
X_comp = tsne.fit_transform(X)
```

```
cancer_comp_df = pd.DataFrame(data=X_comp)
cancer_comp_df['target'] = y
cancer_comp_df
```

	0	1	target
0	8.493168	-9.088554	0
1	8.812732	-8.316459	0
2	10.252729	-10.500170	0
3	12.267102	-12.656333	0
4	2.671253	-2.421643	0
...	...	...	...
173	3.013271	-2.087506	2
174	3.272086	-2.438468	2
175	5.032075	-4.438871	2
176	4.906989	-4.780922	2
177	-3.598375	4.580731	2

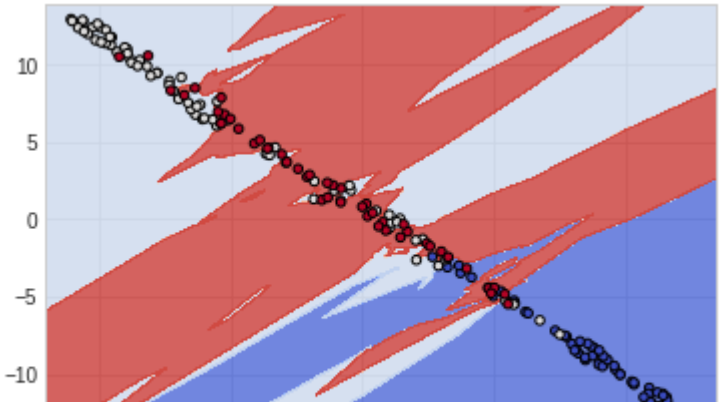
178 rows × 3 columns

```
plt.scatter(X_comp[:, 0], X_comp[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = KNeighborsClassifier()
model.fit(X_comp, y)
predict = model.predict(X_comp)
```

```
xx, yy = make_meshgrid(X_comp[:, 0], X_comp[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_comp[:, 0], X_comp[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



k 최근접 이웃 회귀

- k 최근접 이웃 분류와 마찬가지로 예측에 이웃 데이터 포인트 사용
- 이웃 데이터 포인트의 평균이 예측 결과

보스턴 주택 가격 데이터

```
boston = load_boston()
```

```
boston_df = pd.DataFrame(data=boston.data, columns=boston.feature_names)
boston_df['TARGET'] = boston.target
boston_df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90

506 rows × 14 columns

```
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
boston_train_df = pd.DataFrame(data=X_train, columns=boston.feature_names)
boston_train_df['TARGET'] = y_train
boston_train_df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	2.44668	0.0	19.58	0.0	0.8710	5.272	94.0	1.7364	5.0	403.0	14.7	88.6
1	8.49213	0.0	18.10	0.0	0.5840	6.348	86.1	2.0527	24.0	666.0	20.2	83.4
2	0.38735	0.0	25.65	0.0	0.5810	5.613	95.6	1.7572	2.0	188.0	19.1	359.2
3	0.15936	0.0	6.91	0.0	0.4480	6.211	6.5	5.7209	3.0	233.0	17.9	394.4
4	0.09512	0.0	12.83	0.0	0.4370	6.286	45.0	4.5026	5.0	398.0	18.7	383.2
...	...	...	...	...	...	...	...	...	...	...	...	
399	7.52601	0.0	18.10	0.0	0.7130	6.417	98.3	2.1850	24.0	666.0	20.2	304.2
400	0.05023	35.0	6.06	0.0	0.4379	5.706	28.4	6.6407	1.0	304.0	16.9	394.0
401	0.29916	20.0	6.96	0.0	0.4640	5.856	42.1	4.4290	3.0	223.0	18.6	388.6
402	0.07503	33.0	2.18	0.0	0.4720	7.420	71.9	3.0992	7.0	222.0	18.4	396.9
403	0.05083	0.0	5.19	0.0	0.5150	6.316	38.1	6.4584	5.0	224.0	20.2	389.7

404 rows × 14 columns

```
boston_test_df = pd.DataFrame(data=X_test, columns=boston.feature_names)
boston_test_df['TARGET'] = y_test
boston_test_df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	0.06466	70.0	2.24	0.0	0.400	6.345	20.1	7.8278	5.0	358.0	14.8	368.2
1	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.8
2	37.66190	0.0	18.10	0.0	0.679	6.202	78.7	1.8629	24.0	666.0	20.2	18.8
3	0.77299	0.0	8.14	0.0	0.538	6.495	94.4	4.4547	4.0	307.0	21.0	387.9
4	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0	19.2	396.9
...	...	...	...	...	...	...	...	...	...	...	...	
97	1.00245	0.0	8.14	0.0	0.538	6.674	87.3	4.2390	4.0	307.0	21.0	380.2
98	1.80028	0.0	19.58	0.0	0.605	5.877	79.2	2.4259	5.0	403.0	14.7	227.6
99	5.09017	0.0	18.10	0.0	0.713	6.297	91.8	2.3682	24.0	666.0	20.2	385.0
100	0.18337	0.0	27.74	0.0	0.609	5.414	98.3	1.7554	4.0	711.0	20.1	344.0
101	0.11069	0.0	13.89	1.0	0.550	5.951	93.8	2.8893	5.0	276.0	16.4	396.9

102 rows × 14 columns

```
scaler = StandardScaler()
```

```
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform(X_test)
```

```
model = KNeighborsRegressor()
model.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7185515605294484
평가 데이터 점수: 0.465231418536561
```

```
model = KNeighborsRegressor()
model.fit(X_train_scale, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train_scale, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test_scale, y_test)))
```

```
학습 데이터 점수: 0.8344433361266227
평가 데이터 점수: 0.8263043985776729
```

```
estimator = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor()
)
```

```
cross_validate(
    estimator=estimator,
    X=X, y=y,
    cv=5,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   5 out of   5 | elapsed:    0.0s finished
{'fit_time': array([0.00239539, 0.00340152, 0.00232053, 0.00218725, 0.00213408]),
 'score_time': array([0.00401759, 0.00436521, 0.00282955, 0.00309706, 0.00203395]),
 'test_score': array([0.56089547, 0.61917359, 0.48661916, 0.46986886, 0.23133037])}
```

```
pipe = Pipeline(
    [('scaler', StandardScaler()),
     ('model', KNeighborsRegressor())]
)
```

```
param_grid = [{'model__n_neighbors': [3, 5, 7],
               'model__weights': ['uniform', 'distance'],
               'model__algorithm': ['ball_tree', 'kd_tree', 'brute']}]
```

```
gs = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
gs.fit(X, y)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  90 out of  90 | elapsed:    0.4s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('model',
                                        KNeighborsRegressor(algorithm='auto',
                                                            leaf_size=30,
                                                            metric='minkowski',
                                                            metric_params=None,
                                                            n_jobs=None,
                                                            n_neighbors=5, p=2,
                                                            weights='uniform'))],
                                verbose=False),
             iid='deprecated', n_jobs=2,
             param_grid=[{'model__algorithm': ['ball_tree', 'kd_tree', 'brute'],
                          'model__n_neighbors': [3, 5, 7],
                          'model__weights': ['uniform', 'distance']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

```
gs.best_estimator_
```

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('model',
                 KNeighborsRegressor(algorithm='ball_tree', leaf_size=30,
                                     metric='minkowski', metric_params=None,
                                     n_jobs=None, n_neighbors=7, p=2,
                                     weights='distance'))],
         verbose=False)
```

```
print('GridSearchCV best score: {}'.format(gs.best_score_))
```

```
GridSearchCV best score: 0.4973060611762845
```

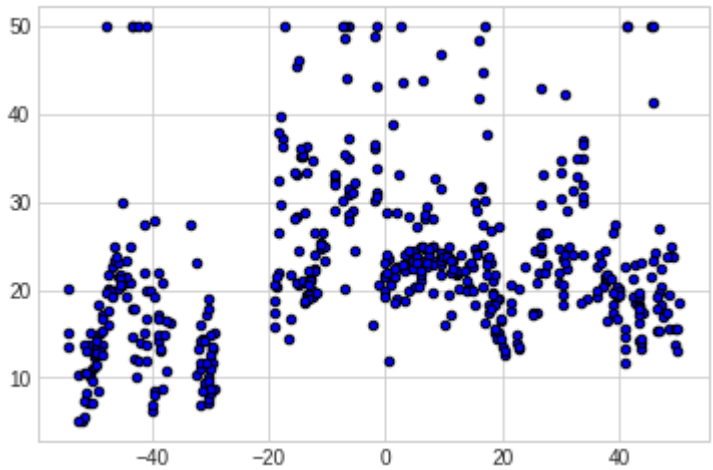
```
tsne = TSNE(n_components=1)
X_comp = tsne.fit_transform(X)
```

```
boston_comp_df = pd.DataFrame(data=X_comp)
boston_comp_df['target'] = y
boston_comp_df
```

	0	target
0	12.986978	24.0
1	-16.254101	21.6
2	-12.456803	34.7
3	-13.418537	33.4
4	-13.687325	36.2
...	...	...
501	1.604652	22.4
502	0.945754	20.6
503	0.147127	23.9
504	0.193567	22.0
505	0.661292	11.9

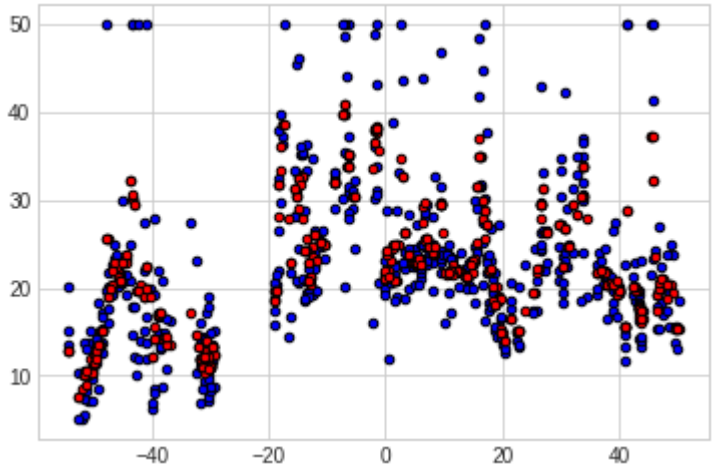
506 rows × 2 columns

```
plt.scatter(X_comp, y, c='b', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = KNeighborsRegressor()
model.fit(X_comp, y)
predict = model.predict(X_comp)
```

```
plt.scatter(X_comp, y, c='b', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
plt.scatter(X_comp, predict, c='r', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



캘리포니아 주택 가격 데이터

```
california = fetch_california_housing()
```

```
california_df = pd.DataFrame(data=california.data, columns=california.feature_names)
california_df['TARGET'] = california.target
california_df
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121

20640 rows × 9 columns

```
X, y = california.data, california.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
california_train_df = pd.DataFrame(data=X_train, columns=california.feature_names)
california_train_df['TARGET'] = y_train
california_train_df
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longit
0	2.7396	7.0	6.548077	1.201923	485.0	4.663462	34.59	-11
1	3.3479	30.0	4.732255	1.125786	4065.0	3.652291	37.64	-12
2	1.3713	23.0	4.329939	1.030550	2091.0	4.258656	36.96	-12
3	4.8100	42.0	4.979499	0.963554	1145.0	2.608200	34.26	-11
4	3.6902	41.0	5.221311	1.040984	1290.0	2.643443	36.99	-12
...	...	...	...	...	...	...	...	
16507	5.1778	26.0	5.097087	0.970874	881.0	2.851133	33.81	-11
16508	5.5840	13.0	7.302128	1.097872	1286.0	2.736170	38.67	-12
16509	12.7823	38.0	7.422481	1.003876	1292.0	2.503876	34.13	-11
16510	1.4423	32.0	4.090753	1.116438	1672.0	2.863014	37.93	-12
16511	7.6740	15.0	5.387879	0.942424	908.0	2.751515	33.86	-11

16512 rows × 9 columns

```
california_test_df = pd.DataFrame(data=X_test, columns=california.feature_names)
california_test_df['TARGET'] = y_test
california_test_df
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	4.3542	34.0	5.578313	0.969880	978.0	2.945783	33.20	-117.
1	4.0119	26.0	7.120000	1.400000	293.0	2.930000	38.92	-122.
2	3.7174	4.0	3.567939	1.172519	983.0	1.500763	33.69	-117.
3	3.1957	29.0	4.355839	1.040146	1978.0	3.609489	33.91	-118.
4	3.8427	35.0	5.471311	1.010246	1196.0	2.450820	33.93	-118.
...	...	...	...	...	...	...	...	
4123	7.0592	16.0	7.434579	1.002336	1663.0	3.885514	33.87	-117.
4124	1.8191	36.0	4.227666	1.063401	1247.0	3.593660	33.93	-118.
4125	1.6531	30.0	3.700357	1.107015	2739.0	3.256837	33.98	-118.
4126	3.6667	47.0	4.339960	1.013917	1445.0	2.872763	34.11	-118.
4127	2.6312	28.0	4.169312	1.058201	891.0	2.357143	33.90	-118.

4128 rows × 9 columns

```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform(X_test)
```

```
model = KNeighborsRegressor()
```

```
model.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.44916063107970566
평가 데이터 점수: 0.1738179980597756
```

```
model = KNeighborsRegressor()
model.fit(X_train_scale, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
print("학습 데이터 점수: {}".format(model.score(X_train_scale, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test_scale, y_test)))
```

```
학습 데이터 점수: 0.7957989340628304
평가 데이터 점수: 0.6780599276783348
```

```
estimator = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor()
)
```

```
cross_validate(
    estimator=estimator,
    X=X, y=y,
    cv=5,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   5 out of   5 | elapsed:    2.3s finished
{'fit_time': array([0.03975701, 0.04289508, 0.03692508, 0.03664112, 0.03614807]),
 'score_time': array([0.56828904, 0.46674418, 0.4789927 , 0.57057905, 0.38986254]),
 'test_score': array([0.47879396, 0.4760079 , 0.57624554, 0.50259828, 0.57228584])}
```

```
pipe = Pipeline(
    [('scaler', StandardScaler()),
     ('model', KNeighborsRegressor())]
)
```

```
param_grid = [{'model__n_neighbors': [3, 5, 7],
               'model__weights': ['uniform', 'distance'],
               'model__algorithm': ['ball_tree', 'kd_tree', 'brute']}]
```

```
gs = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    verbose=True
)
```

```
gs.fit(X, y)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:   22.8s
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:691: UserWar
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=2)]: Done  90 out of  90 | elapsed:   58.1s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('model',
                                        KNeighborsRegressor(algorithm='auto',
                                                            leaf_size=30,
                                                            metric='minkowski',
                                                            metric_params=None,
                                                            n_jobs=None,
                                                            n_neighbors=5, p=2,
                                                            weights='uniform'))],
                                verbose=False),
             iid='deprecated', n_jobs=2,
             param_grid=[{'model__algorithm': ['ball_tree', 'kd_tree', 'brute'],
                          'model__n_neighbors': [3, 5, 7],
                          'model__weights': ['uniform', 'distance']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

```
gs.best_estimator_
```

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('model',
                 KNeighborsRegressor(algorithm='ball_tree', leaf_size=30,
                                     metric='minkowski', metric_params=None,
                                     n_jobs=None, n_neighbors=7, p=2,
                                     weights='distance'))],
         verbose=False)
```

```
print('GridSearchCV best score: {}'.format(gs.best_score_))
```

```
GridSearchCV best score: 0.5376515274379832
```

```
tsne = TSNE(n_components=1)
```

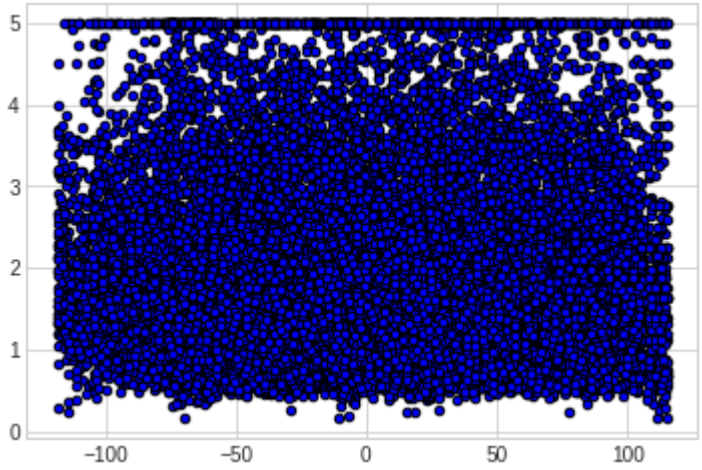
```
X_comp = tsne.fit_transform(X)
```

```
boston_comp_df = pd.DataFrame(data=X_comp)
boston_comp_df['target'] = y
boston_comp_df
```

	0	target
0	-9.103486	4.526
1	-86.075554	3.585
2	-74.965141	3.521
3	-65.984032	3.413
4	-65.434853	3.422
...	...	...
20635	36.445869	0.781
20636	-57.132732	0.771
20637	56.534737	0.923
20638	21.282549	0.847
20639	-26.039883	0.894

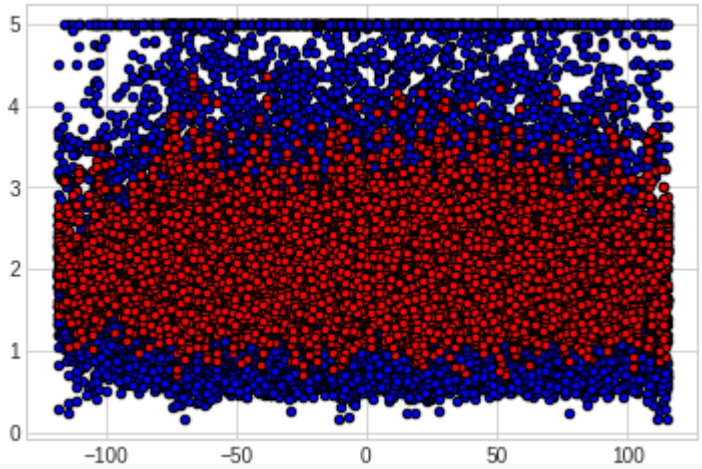
20640 rows × 2 columns

```
plt.scatter(X_comp, y, c='b', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = KNeighborsRegressor()
model.fit(X_comp, y)
predict = model.predict(X_comp)
```

```
plt.scatter(X_comp, y, c='b', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
plt.scatter(X_comp, predict, c='r', cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```


5
4
3
2
1
0
−100
−50
0
50
100