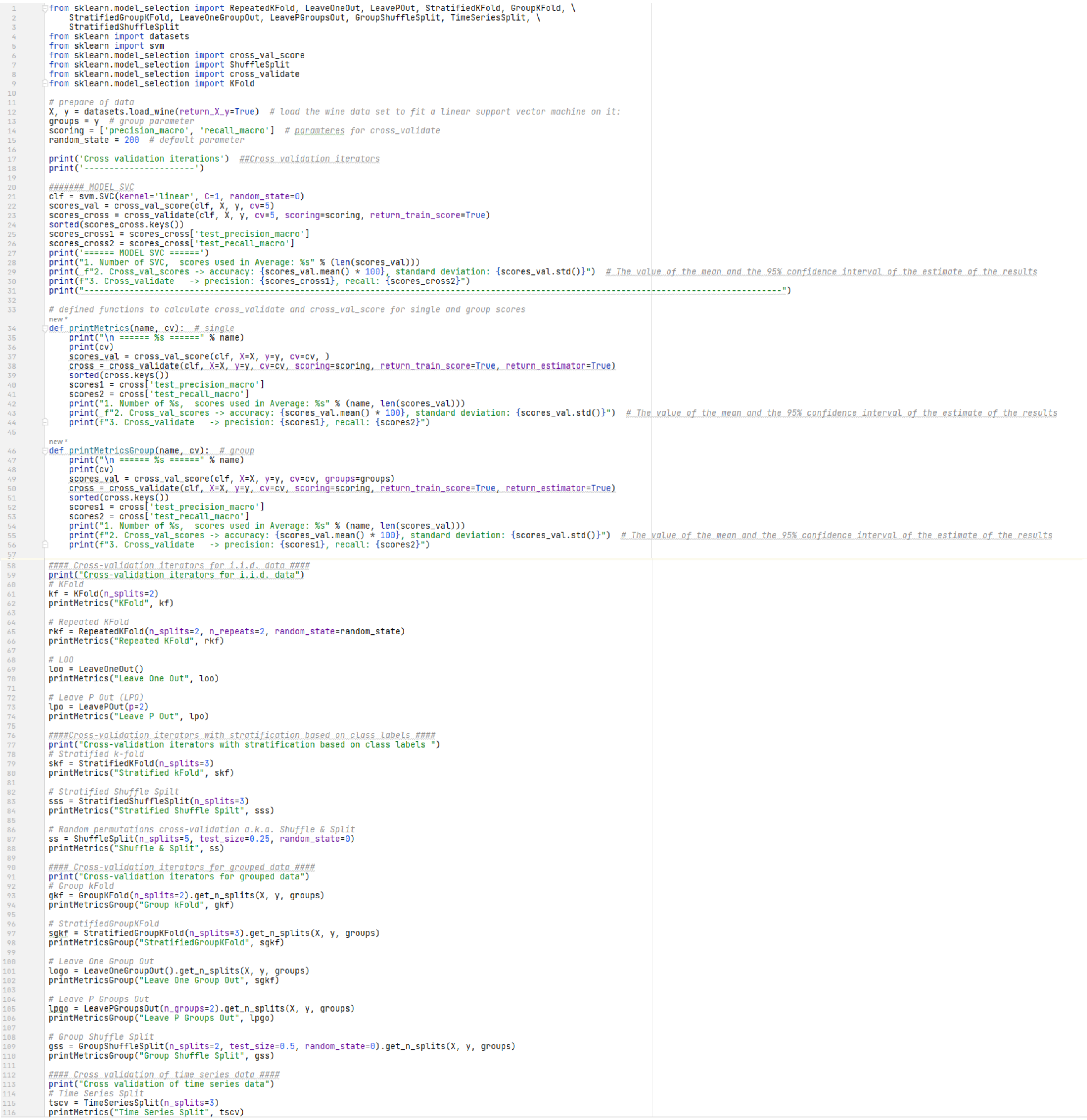
Sieci neuronowe i sztuczna inteligencja – laboratorium 3

24.03.2023

Monika Błyszcz, 236623

Zadanie wykonano na domyślnej bazie danych dataset.load_wine, którą można bezpośrednio zaimplementować do Pythona ze sklearn. Bazę przygotowano za pomocą komendy: **X,y = datasets.load_wine(return_X_y=True)**

Następnie przetestowano wszystkie typy walidacji krzyżowej *(łącznie 13, wymienione na uzyskanych wynikach na następnej stronie)* i porównano używając następujących modeli oszacowania jak: CROSS_VAL_SCORE *(pobiera średnią z foldów walidacji krzyżowej)* oraz CROSS_VALIDATE: *(który pozwala określić wiele metryk do oceny oraz zwraca zestawienie zawierające czasy dopasowania, czasy punktacji (i opcjonalnie wyniki treningowe oraz dopasowane estymatory) oprócz wyniku testu.).* Poniżej przedstawiono kod implementujący opisane powyżej działania

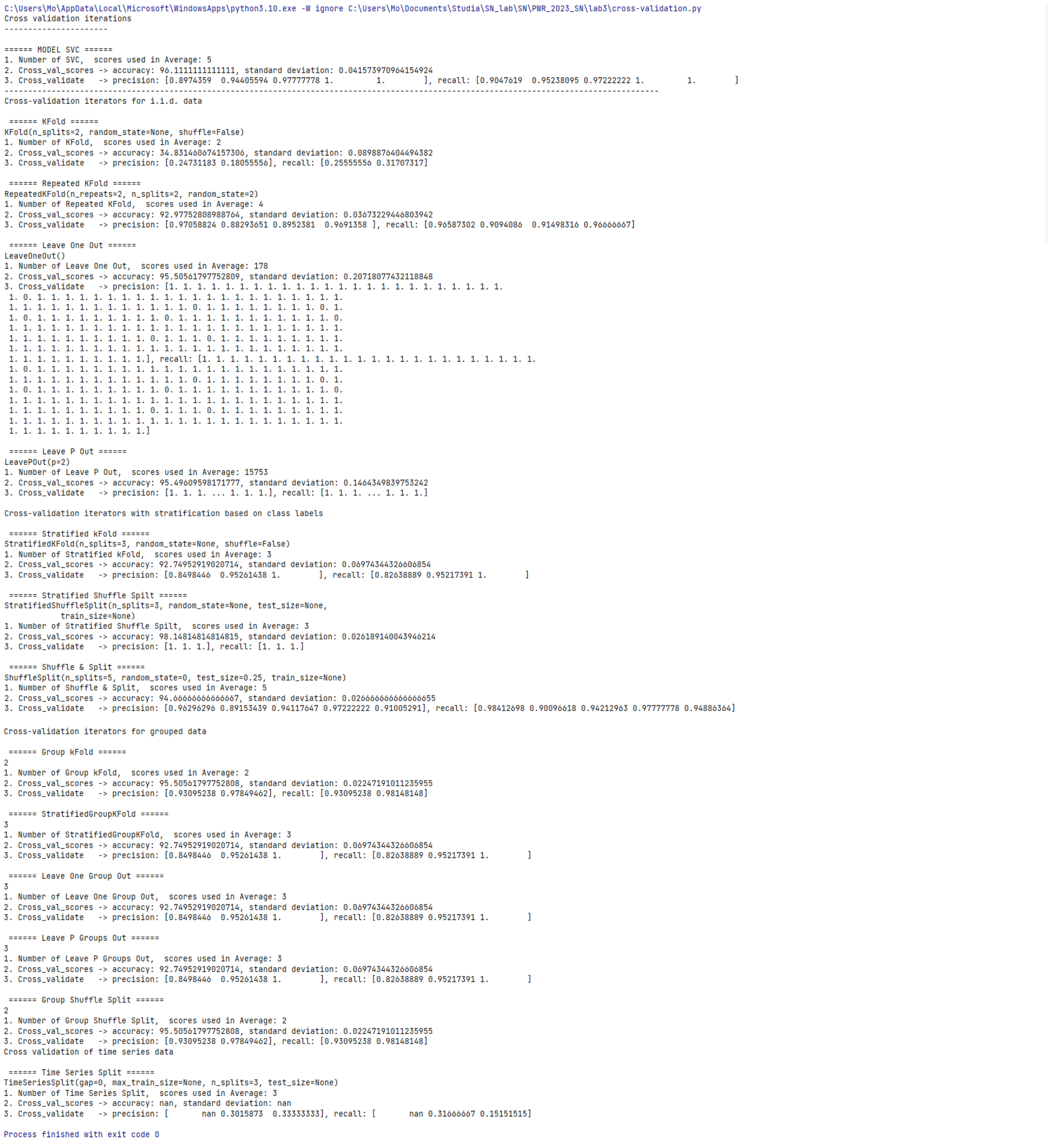


```python
from sklearn.model_selection import RepeatedKFold, LeaveOneOut, LeavePOut, StratifiedKFold, GroupKFold, \
    StratifiedGroupKFold, LeaveOneGroupOut, LeavePGroupsOut, GroupShuffleSplit, TimeSeriesSplit, \
    StratifiedShuffleSplit
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold

# prepare of data
X, y = datasets.load_wine(return_X_y=True)  # load the wine data set to fit a linear support vector machine on it:
groups = y  # group parameter
scoring = ['precision_macro', 'recall_macro']  # paramteres for cross_validate
random_state = 200  # default parameter

print('Cross validation iterations')  ##Cross validation iterators
print('----------------------')

####### MODEL SVC
clf = svm.SVC(kernel='linear', C=1, random_state=0)
scores_val = cross_val_score(clf, X, y, cv=5)
scores_cross = cross_validate(clf, X, y, cv=5, scoring=scoring, return_train_score=True)
sorted(scores_cross.keys())
scores_cross1 = scores_cross['test_precision_macro']
scores_cross2 = scores_cross['test_recall_macro']
print('====== MODEL SVC ======')
print("1. Number of SVC,  scores used in Average: %s" % (len(scores_val)))
print( f"2. Cross_val_scores -> accuracy: {scores_val.mean() * 100}, standard deviation: {scores_val.std()}")  # The value of the mean and the 95% confidence interval of the estimate of the results
print(f"3. Cross_validate   -> precision: {scores_cross1}, recall: {scores_cross2}")
print("------------------------------------------------------------------------------------------------------------------------------------------")

# defined functions to calculate cross_validate and cross_val_score for single and group scores
def printMetrics(name, cv):  # single
    print("\n ====== %s ======" % name)
    print(cv)
    scores_val = cross_val_score(clf, X=X, y=y, cv=cv, )
    cross = cross_validate(clf, X=X, y=y, cv=cv, scoring=scoring, return_train_score=True, return_estimator=True)
    sorted(cross.keys())
    scores1 = cross['test_precision_macro']
    scores2 = cross['test_recall_macro']
    print("1. Number of %s,  scores used in Average: %s" % (name, len(scores_val)))
    print( f"2. Cross_val_scores -> accuracy: {scores_val.mean() * 100}, standard deviation: {scores_val.std()}")  # The value of the mean and the 95% confidence interval of the estimate of the results
    print(f"3. Cross_validate   -> precision: {scores1}, recall: {scores2}")

def printMetricsGroup(name, cv):  # group
    print("\n ====== %s ======" % name)
    print(cv)
    scores_val = cross_val_score(clf, X=X, y=y, cv=cv, groups=groups)
    cross = cross_validate(clf, X=X, y=y, cv=cv, scoring=scoring, return_train_score=True, return_estimator=True)
    sorted(cross.keys())
    scores1 = cross['test_precision_macro']
    scores2 = cross['test_recall_macro']
    print("1. Number of %s,  scores used in Average: %s" % (name, len(scores_val)))
    print(f"2. Cross_val_scores -> accuracy: {scores_val.mean() * 100}, standard deviation: {scores_val.std()}")  # The value of the mean and the 95% confidence interval of the estimate of the results
    print(f"3. Cross_validate   -> precision: {scores1}, recall: {scores2}")

#### Cross-validation iterators for i.i.d. data ####
print("Cross-validation iterators for i.i.d. data")
# KFold
kf = KFold(n_splits=2)
printMetrics("KFold", kf)

# Repeated KFold
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=random_state)
printMetrics("Repeated KFold", rkf)

# LOO
loo = LeaveOneOut()
printMetrics("Leave One Out", loo)

# Leave P Out (LPO)
lpo = LeavePOut(p=2)
printMetrics("Leave P Out", lpo)

####Cross-validation iterators with stratification based on class labels ####
print("Cross-validation iterators with stratification based on class labels ")
# Stratified k-fold
skf = StratifiedKFold(n_splits=3)
printMetrics("Stratified kFold", skf)

# Stratified Shuffle Spilt
sss = StratifiedShuffleSplit(n_splits=3)
printMetrics("Stratified Shuffle Spilt", sss)

# Random permutations cross-validation a.k.a. Shuffle & Split
ss = ShuffleSplit(n_splits=5, test_size=0.25, random_state=0)
printMetrics("Shuffle & Split", ss)

#### Cross-validation iterators for grouped data ####
print("Cross-validation iterators for grouped data")
# Group kFold
gkf = GroupKFold(n_splits=2).get_n_splits(X, y, groups)
printMetricsGroup("Group kFold", gkf)

# StratifiedGroupKFold
sgkf = StratifiedGroupKFold(n_splits=3).get_n_splits(X, y, groups)
printMetricsGroup("StratifiedGroupKFold", sgkf)

# Leave One Group Out
logo = LeaveOneGroupOut().get_n_splits(X, y, groups)
printMetricsGroup("Leave One Group Out", sgkf)

# Leave P Groups Out
lpgo = LeavePGroupsOut(n_groups=2).get_n_splits(X, y, groups)
printMetricsGroup("Leave P Groups Out", lpgo)

# Group Shuffle Split
gss = GroupShuffleSplit(n_splits=2, test_size=0.5, random_state=0).get_n_splits(X, y, groups)
printMetricsGroup("Group Shuffle Split", gss)

#### Cross validation of time series data ####
print("Cross validation of time series data")
# Time Series Split
tscv = TimeSeriesSplit(n_splits=3)
printMetrics("Time Series Split", tscv)
```

Dla każdego validatora zwrócono następujące parametry:

- cross_validate: precision, recall

- cross_val_score: accuracy, std, number of used in average



```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe -W ignore C:\Users\Mo\Documents\Studia\SN_lab\SN\PWR_2023_SN\lab3\cross-validation.py
Cross validation iterations
----------------------

====== MODEL SVC ======
1. Number of SVC,  scores used in Average: 5
2. Cross_val_scores -> accuracy: 96.1111111111111, standard deviation: 0.041573970964154924
3. Cross_validate   -> precision: [0.8974359  0.94405594 0.97777778 1.         1.        ], recall: [0.9047619  0.95238095 0.97222222 1.         1.        ]
-------------------------------------------------------------------------------------------------------------------------------------
Cross-validation iterators for i.i.d. data

 ====== KFold ======
KFold(n_splits=2, random_state=None, shuffle=False)
1. Number of KFold,  scores used in Average: 2
2. Cross_val_scores -> accuracy: 34.831460674157306, standard deviation: 0.0898876404494382
3. Cross_validate   -> precision: [0.24731183 0.18055556], recall: [0.25555556 0.31707317]

 ====== Repeated KFold ======
RepeatedKFold(n_repeats=2, n_splits=2, random_state=2)
1. Number of Repeated KFold,  scores used in Average: 4
2. Cross_val_scores -> accuracy: 92.97752808988764, standard deviation: 0.03673229446803942
3. Cross_validate   -> precision: [0.97058824 0.88293651 0.8952381  0.9691358 ], recall: [0.96587302 0.9094086  0.91498316 0.96666667]

 ====== Leave One Out ======
LeaveOneOut()
1. Number of Leave One Out,  scores used in Average: 178
2. Cross_val_scores -> accuracy: 95.50561797752809, standard deviation: 0.20718077432118848
3. Cross_validate   -> precision: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.], recall: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

 ====== Leave P Out ======
LeavePOut(p=2)
1. Number of Leave P Out,  scores used in Average: 15753
2. Cross_val_scores -> accuracy: 95.49609598171777, standard deviation: 0.1464349839753242
3. Cross_validate   -> precision: [1. 1. 1. ... 1. 1. 1.], recall: [1. 1. 1. ... 1. 1. 1.]

Cross-validation iterators with stratification based on class labels

 ====== Stratified kFold ======
StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
1. Number of Stratified kFold,  scores used in Average: 3
2. Cross_val_scores -> accuracy: 92.74952919020714, standard deviation: 0.06974344326606854
3. Cross_validate   -> precision: [0.8498446  0.95261438 1.        ], recall: [0.82638889 0.95217391 1.        ]

 ====== Stratified Shuffle Spilt ======
StratifiedShuffleSplit(n_splits=3, random_state=None, test_size=None,
            train_size=None)
1. Number of Stratified Shuffle Spilt,  scores used in Average: 3
2. Cross_val_scores -> accuracy: 98.14814814814815, standard deviation: 0.026189140043946214
3. Cross_validate   -> precision: [1. 1. 1.], recall: [1. 1. 1.]

 ====== Shuffle & Split ======
ShuffleSplit(n_splits=5, random_state=0, test_size=0.25, train_size=None)
1. Number of Shuffle & Split,  scores used in Average: 5
2. Cross_val_scores -> accuracy: 94.66666666666667, standard deviation: 0.026666666666666655
3. Cross_validate   -> precision: [0.96296296 0.89153439 0.94117647 0.97222222 0.91005291], recall: [0.98412698 0.90096618 0.94212963 0.97777778 0.94886364]

Cross-validation iterators for grouped data

 ====== Group kFold ======
2
1. Number of Group kFold,  scores used in Average: 2
2. Cross_val_scores -> accuracy: 95.50561797752808, standard deviation: 0.02247191011235955
3. Cross_validate   -> precision: [0.93095238 0.97849462], recall: [0.93095238 0.98148148]

 ====== StratifiedGroupKFold ======
3
1. Number of StratifiedGroupKFold,  scores used in Average: 3
2. Cross_val_scores -> accuracy: 92.74952919020714, standard deviation: 0.06974344326606854
3. Cross_validate   -> precision: [0.8498446  0.95261438 1.        ], recall: [0.82638889 0.95217391 1.        ]

 ====== Leave One Group Out ======
3
1. Number of Leave One Group Out,  scores used in Average: 3
2. Cross_val_scores -> accuracy: 92.74952919020714, standard deviation: 0.06974344326606854
3. Cross_validate   -> precision: [0.8498446  0.95261438 1.        ], recall: [0.82638889 0.95217391 1.        ]

 ====== Leave P Groups Out ======
3
1. Number of Leave P Groups Out,  scores used in Average: 3
2. Cross_val_scores -> accuracy: 92.74952919020714, standard deviation: 0.06974344326606854
3. Cross_validate   -> precision: [0.8498446  0.95261438 1.        ], recall: [0.82638889 0.95217391 1.        ]

 ====== Group Shuffle Split ======
2
1. Number of Group Shuffle Split,  scores used in Average: 2
2. Cross_val_scores -> accuracy: 95.50561797752808, standard deviation: 0.02247191011235955
3. Cross_validate   -> precision: [0.93095238 0.97849462], recall: [0.93095238 0.98148148]
Cross validation of time series data

 ====== Time Series Split ======
TimeSeriesSplit(gap=0, max_train_size=None, n_splits=3, test_size=None)
1. Number of Time Series Split,  scores used in Average: 3
2. Cross_val_scores -> accuracy: nan, standard deviation: nan
3. Cross_validate   -> precision: [       nan 0.3015873  0.33333333], recall: [       nan 0.31666667 0.15151515]

Process finished with exit code 0
```

Analizując kod możemy zauważyć, że w ogólnym ujęciu najlepsze parametry dla obu metod uzyskał Shuffle – Spilt (precision: 0,9629, re-call: 0,9841, accuracy: 0,9466). Natomiast wyszczególniając na konkretne metody to:

- Najlepsze wyniki dla cross_validate uzyskano dla: Shuffle-Spilt (precision: 0,9629, re-call: 0,9841)
- Najlepsze wyniki dla cross_val_score uzyskano dla: Stratified Shuffle Spilt (accuracy: 98,14).

Na podstawie wyników można wywnioskować, że lepszą metodą jest cross_validate, ponieważ zwraca więcej metryk oceny, co pozwala na dokładniejsze sprawdzenie strategii krzyżowej. Natomiast pod względem wyników lepszy jest cross_val_score bo uzyskuje wyższe wartości dokładności.