# Cross-Validation implementation

# There are different Cross-Validation techniques

- Hold-out cross-validation
- K-folds cross-validation
- Leave-one-out cross-validation
- Leave-p-out cross-validation
- Stratified K-folds cross-validation
- Repeated K-folds cross-validation
- Group K-Fold Cross-Validation
- Time series CV cross-validation

Hold-Out based Validation

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_
```

K-folds cross-validation

In [2]:
```
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)

for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
```

- Leave-one-out cross-validation

In [3]:
```python
import numpy as np
from sklearn.model_selection import LeaveOneOut

X = np.array([[1, 2], [3, 4]])
y = np.array([1, 2])
loo = LeaveOneOut()

for train_index, test_index in loo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [1] TEST: [0]
TRAIN: [0] TEST: [1]
```

- Leave-p-out cross-validation

In [4]:
```python
import numpy as np
from sklearn.model_selection import LeavePOut

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 3, 4])
lpo = LeavePOut(2)

for train_index, test_index in lpo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [2 3] TEST: [0 1]
TRAIN: [1 3] TEST: [0 2]
TRAIN: [1 2] TEST: [0 3]
TRAIN: [0 3] TEST: [1 2]
TRAIN: [0 2] TEST: [1 3]
TRAIN: [0 1] TEST: [2 3]
```

- Stratified k-Fold cross-validation

In [5]:
```
import numpy as np
from sklearn.model_selection import StratifiedKFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
skf = StratifiedKFold(n_splits=2)

for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
```

- Repeated k-Fold cross-validation

In [6]:
```
import numpy as np
from sklearn.model_selection import RepeatedKFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=42)

for train_index, test_index in rkf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [0 2] TEST: [1 3]
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
TRAIN: [1 3] TEST: [0 2]
```

- Group K-Fold Cross-Validation

In [7]:
```
from sklearn.model_selection import GroupKFold
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Generate some sample data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_

# Generate random groups
import numpy as np
groups = np.random.randint(0, 5, size=len(X))

# Define the number of folds
n_splits = 5

# Initialize the GroupKFold object
gkf = GroupKFold(n_splits=n_splits)

# Initialize a classifier (you can use any model of your choice)
classifier = LogisticRegression()

# Iterate over the splits
for train_index, test_index in gkf.split(X, y, groups=groups):
    # Split the data into train and test sets based on the current split
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
```

```
Accuracy: 0.863849765258216
Accuracy: 0.8883495145631068
Accuracy: 0.8578680203045685
Accuracy: 0.8808290155440415
Accuracy: 0.8743455497382199
```

Time series CV cross-validation

In [8]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import TimeSeriesSplit
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Generate synthetic time series data
np.random.seed(42)
dates = pd.date_range(start='2022-01-01', periods=100, freq='D')
values = np.sin(np.arange(100) * np.pi / 10) + np.random.randn(100) * 0.1  #
df = pd.DataFrame({'Date': dates, 'Value': values})

# Define the number of splits (number of folds)
n_splits = 5

# Initialize the TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits=n_splits)

# Initialize a model (Decision Tree Regressor)
model = DecisionTreeRegressor(max_depth=3)  # Limiting depth for simplicity

# Initialize lists to store evaluation metrics
mse_scores = []

# Perform time series cross-validation
for train_index, test_index in tscv.split(df):
    # Split the data into train and test sets based on the current split
    train_data, test_data = df.iloc[train_index], df.iloc[test_index]

    # Prepare features and target variables
    X_train, y_train = train_data[['Date']], train_data['Value']
    X_test, y_test = test_data[['Date']], test_data['Value']

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate mean squared error (you can use any other evaluation metric)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)

    # Optionally, you can print or store other evaluation metrics as needed
    print("Mean Squared Error:", mse)

# Calculate the mean of the evaluation metric scores
mean_mse = np.mean(mse_scores)
print("Mean MSE:", mean_mse)
```

```
Mean Squared Error: 1.07780361818398
Mean Squared Error: 1.0416604122069493
Mean Squared Error: 0.5946104630946154
Mean Squared Error: 1.9091032877040826
Mean Squared Error: 0.5664471358732494
Mean MSE: 1.0379249834125754
```

In [ ]:

```

```