Cross-Validation (CV) → Is a technique for evaluating a ML model and testing its performance. It helps to compare and select an appropiate model for the specific predictive modeling problem.

There are a lot of different techniques that may be used to cross-validate a model. Still, all of them have a similar algorithm:

1. Divide the dataset into two parts: one for training, other for testing.
2. Train the model in the training set.
3. Validate the model on the test set.
4. Repeat 1-3 steps a couple of times. This number depends on the CV method that you are using.

Cross-Validation techniques

Hold-out cross-validation → Is the simplest and most common technique. We use hold-out method on large datasets as it requires training the model only once.
The fact that we test our model only once might be a bottleneck for this method. Due to this, the result obtained by hold-out technique may be considered inaccurate.

Algorithm:

1. Divide the dataset into two parts: the training set and the test set. Usually 80% of the dataset goes to the training set and 20% to the test set but you may choose any splitting that suits you better.
2. Train the model on the training set.
3. Validate on the test set.
4. Save the result of the validation.

Implementation:

```
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_state=111)
```

k-Fold Cross-Validation → Is a technique that minimizes the disadvantages of hold-out method. k-Fold introduces a new way of splitting the dataset which helps to overcome the "test only once bottleneck".
k-Fold gives a more stable and trustworthy result as previous method since training and testing is performed on several different parts of the dataset.
Still, k-Fold method has a disadvantage. Increasing k results in training more models and the training process may be really expensive and time-consuming.

Algorithm:

1. Pick a number of folds - k. Usually, k is 5 to 10 but you can choose any number which is less than the dataset's length.
2. Split the dataset into k equal (if possible) parts (they are called folds)
3. Choose k-1 folds which will be the training set. The remaining fold will be the test set.
4. Train the model on the training set. On each iteration of cross-validation, you must train a new model independently of the model trained in the previous iteration.
5. Validate on the test set.
6. Save the result of the validation.
7. Repeat steps 3-6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

Implementation:

```
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]
```

Leave-one-out cross-validation (LOOCV) → Is an extreme case of k-Fold CV where k is equal to n, where n is the number of samples in the dataset.
The greatest advantage of LOOCV is that it doesn't waste much data. LOOCV is more computationally expensive than k-Fold, it make take plenty of time to cross-validate the model using LOOCV.
Thus, the Data Science community has a general rule based on empirical evidence and different researches, which suggest that 5 or 10 fold cross-validation should be preferred over LOOCV.

Implementation:

```
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]
```

Leave-p-out cross-validation (LpOCV) → Is similar to LOOCV as it creates all the possible training and test sets by using p samples as the test set.
LpOCV has all disadvantages of the LOOCV, but, nevertheless, it's as robust as LOOCV

Algorithm:

1. Choose p samples from the dataset which will be the test set.
2. The remaining n-p samples will be the training set.
3. Train the model on the training set. On each iteration, a new model must be trained.
4. Validate on the test set.
5. Save the result of the validation.
6. Repeat steps 2-5 C_p^n times.
7. To get the final score average the results that you got on step 5.