Pseudocódigos

```
function: get_confusion_matrix
  inputs:
   - y_real: array of size N
   - y_predicted: array of size N

  code:
    predicted_true_real_true <- sum((y_predicted .== 1) .& (y_real .== 1))
    predicted_true_real_false <- sum((y_predicted .== 1) .& (y_real .== 0))
    predicted_false_real_true <- sum((y_predicted .== 0) .& (y_real .== 1))
    predicted_false_real_false <- sum((y_predicted .== 0) .& (y_real .== 0))

    confusion_matrix = [
                        [predicted_true_real_true, predicted_false_real_true]
                         [predicted_true_real_false, predicted_false_real_false]
                       ]

  outputs:
    - confusion_matrix
```

```
function: get_accuray_score
  inputs:
   - confusion_matrix: 2x2 matrix

  code:
    right_predictions <- sum(diagonal(confusion_matrix))
    total <- sum(sum(matrix)) # sum over axis 0, then axis 1
    accuracy_score <- right_predictions / total

  outputs:
    - accuracy_score
```

```
function: get_precision_score

  inputs:
   - confusion_matrix: 2x2 matrix of integers

  code:
    true_positives <- confusion_matrix[0][0]
    false_positives <- confusion_matrix[1][0]
    error <- 0.0000000001 # to avoid division for 0
    precision_score <- true_positives / (true_positives +  false_positives + error)

  outputs:
    - precision_score
```

```
function: get_recall_score

  inputs:
   - confusion_matrix: 2x2 matrix of integers

  code:
    true_positives <- confusion_matrix[0][0]
    false_negatives <- confusion_matrix[0][1]
    error <- 0.0000000001 # to avoid division for 0
    recall_score <- true_positives / (true_positives + false_negatives + error)

  outputs:
    - recall_score
```

```
function: get_f1_score
  inputs:
   - confusion_matrix: 2x2 matrix of integers

  code:
    precision_score <- get_precision_score(confusion_matrix)
    recall_score <- get_recall_score(confusion_matrix)
    error <- 0.0000000001 # to avoid division for 0
    f1_score <- 2 * (
              (precision_score * recall_score) /
              (precision_score + recall_score + error)
            )
  outputs:
    - f1_score
```

```
function: generate_folds
  input:
    - y_values: array of size N
    - number_of_folds: integer

  code:
    indices_with_false_label = []
    indices_with_true_label = []
    for index <- 0 to N-1 then
      if y_values[index] == 0 then
        indices_with_false_label->append(index)
      else
        indices_with_true_label->append(index)
      endif
    endfor

    indices_with_false_label <- shuffle(indices_with_false_label)
    indices_with_true_label <- shuffle(indices_with_false_label)
    folds <- get_list_of_empty_folds(number_of_folds)

    for _ <- 0 to int(N / number_folds) then
      for index_fold <- 0 to number_folds then
        if size(indices_with_false_label) > 0 & size(indices_with_true_label) > 0 then
          index_false_label = indices_with_false_label->pop(0)
          index_true_label = indices_with_true_label->pop(0)
          folds[index_fold]->extend([index_false_label, index_true_label])
        else
          return folds
        endif
      endfor
    endfor

  outputs:
    - folds: list of folds, with each fold containing dataset indices
```

```
function: generate_indices_for_splits
  input:
    - number_of_folds: integer

  code:
    indices_for_splits <- []
    for index_fold_test <- 0 to number_of_folds - 1 then
      folds_for_training <- []
      for index_fold_train <- 0 to number_of_folds -1 then
        if index_fold_train != index_fold_test then
          folds_for_training->append(index_fold_train)
        endif
        indices_for_splits->append([folds_for_training, index_fold_test])
      endfor
    endfor

  outputs:
    - indices_for_splits: list containing combinations of training folds and
                          test folds indices
```

```
function: split
  input:
    - x_values: matrix of dimension MxN (M number of observation in
dataset, N number of independent variables in dataset)
    - y_values: array of size N
    - number_of_folds: integer

  code:
    folds <- generate_folds(y_values, number_folds)
    indices_for_splits <- generate_indices_for_splits(number_of_folds)
    indexes_with_false_label <- []

    splits <- []
    for index_split <- 0 to size(splits) then
      train_indexes_folds <- splits[index_split][0]
      x_train <- x_values[folds[train_index_folds]]
      y_train <- y_values[folds[train_index_folds]]

      test_index_fold <- splits[index_split][1]
      x_test <- x_values[folds[test_index_fold]]
      y_test <- y_values[folds[test_index_fold]]

      splits.append([x_train, x_test, y_train, y_test])
    endfor

  outputs:
    - splits: disjoint folds containing data for training and testing
```

```
function: get_best_parameters
  input:
    - classifier: a scikit-learn class for classification
    - parameters_grid: hyperparameter options map for classifier
    - x_values: matrix of dimension MxN (M number of observation in
dataset,
                  number of independent variables in dataset)
    - y_values: array of size N
    - number_of_folds: integer
    - number_of_parameters_combinations: integer

  code:
    parameters_combination <- get_random_combinations_of_parameters(
                              parameters_grid,
                              number_of_parameters_combinations
                             )
    best_parameters <- none
    best_mean_f1_score <- 0

    for index_combination <- 0 to size(parameters_combination) then
      parameters <- parameters_combinations[index_combination]
      scores = []
      splits_of_folds <- split(x_values, y_values, number_of_folds)
      for index_split <- 0 to size(splits_of_folds) then
        x_train <- splits_of_folds[index_split][0]
        x_test <- splits_of_folds[index_split][1]
        y_train <- splits_of_folds[index_split][2]
        y_test <- splits_of_folds[index_split][3]
        test_classifier <- classifier(parameters)
        test_classifier->fit(x_train, y_train)
        test_predictions <- test_classifier->predict(x_test)
        test_confusion_matrix <- get_confusion_matrix(y_test,validation_predictions)
        test_f1_score <- get_f1_score(validation_confusion_matrix)
        scores->append(test_f1_score)
      endfor

      mean_f1_score <- mean(scores)
      if mean_f1_score > best_mean_f1_score then
        best_mean_f1_score <- mean_f1_score
        best_parameters <- parameters
      endif
    endfor

  outputs:
    - best_parameters
```

```
function: run_cross_validation
  input:
    - classifier: a scikit-learn class for classification
    - parameters_grid: hyperparameter options map for classifier
    - x_values: matrix of dimension MxN (M number of observation in dataset,
                number of independent variables in dataset)
    - y_values: array of size N
    - number_of_folds: integer
    - number_of_parameters_combinations: integer

  code:
    splits_of_folds <- split(x_values, y_values, number_of_folds)

    for index_split <- 0 to size(splits_of_folds) then
      x_train <- splits_of_folds[index_split][0]
      x_test <- splits_of_folds[index_split][1]
      y_train <- splits_of_folds[index_split][2]
      y_test <- splits_of_folds[index_split][3]
      best_parameters <- get_best_parameters(classifier,
                         parameters_grids,
                         number_of_folds-1,
                         number_of_parameters_combinations,
                         x_train,
                         y_train
                        )

      best_classifier <- classifier(best_parameters)
      best_classifier->fit(x_train, y_train)

      train_predictions <- best_classifier->predict(x_train)
      train_confusion_matrix <- get_confusion_matrix(y_train, train_predictions)
      train_accuracy <- get_accuracy_score(train_confusion_matrix)
      train_precision <- get_precision_score(train_confusion_matrix)
      train_recall <- get_recall_score(train_confusion_matrix)
      train_f1_score <- get_f1_score(train_confusion_matrix)

      test_predictions <- best_classifier->predict(x_test)
      test_confusion_matrix <- get_confusion_matrix(y_test,validation_predictions)
      test_accuracy <- get_accuracy_score(validation_confusion_matrix)
      test_precision <- get_precision_score(validation_confusion_matrix)
      test_recall <- get_recall_score(validation_confusion_matrix)
      test_f1_score <- get_f1_score(validation_confusion_matrix)

      save_to_csv_all_metrics()
    endfor
```

```
outputs:
```