```python
# ------- BEFORE STARTING - SOME BASIC TIPS
# You can add a comment within a Python file by using a hashtag '#'
# Anything that comes after the hashtag on the same line, will be considered
# a comment and won't be executed as code by the Python interpreter.

# --- 1) IMPORTING PACKAGES
# The first thing you should always do in a Python file is to import any
# packages that you will need within the file. This should always go at the top
# of the file
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler

# --- 2) DEFINE GLOBAL CONSTANTS
# Constants are variables that should remain the same througout the entire running
# of the module. You should define these after the imports at the top of the file.
# You should give global constants a name and ensure that they are in all upper
# case, such as: UPPER_CASE

# K is used to define the number of folds that will be used for cross-validation
K = 10

# Split defines the % of data that will be used in the training sample
# 1 - SPLIT = the % used for testing
SPLIT = 0.75

# --- 3) ALGORITHM CODE
# Next, we should write our code that will be executed when a model needs to be
# trained. There are many ways to structure this code and it is your choice
# how you wish to do this. The code in the 'module_helper.py' file will break
# the code down into independent functions, which is 1 option.
# Include your algorithm code in this section below:

# Load data
def load_data(path: str = "/path/to/csv/"):
    """
    This function takes a path string to a CSV file and loads it into
    a Pandas DataFrame.

    :param      path (optional): str, relative path of the CSV file

    :return     df: pd.DataFrame
    """

    df = pd.read_csv(f"{path}")
    df.drop(columns=["Unnamed: 0"], inplace=True, errors='ignore')
    return df

# Create target variable and predictor variables
def create_target_and_predictors(
    data: pd.DataFrame = None,
    target: str = "estimated_stock_pct"
):
    """
    This function takes in a Pandas DataFrame and splits the columns
    into a target column and a set of predictor variables, i.e. X & y.
    These two splits of the data will be used to train a supervised
    machine learning model.

    :param      data: pd.DataFrame, dataframe containing data for the
                      model
    :param      target: str (optional), target variable that you want to predict

    :return     X: pd.DataFrame
                y: pd.Series
    """
    # Check to see if the target variable is present in the data
    if target not in data.columns:
        raise Exception(f"Target: {target} is not present in the data")

    X = data.drop(columns=[target])
    y = data[target]
    return X, y

# Train algorithm
def train_algorithm_with_cross_validation(
    X: pd.DataFrame = None,
    y: pd.Series = None
):
    """
    This function takes the predictor and target variables and
    trains a Random Forest Regressor model across K folds. Using
    cross-validation, performance metrics will be output for each
    fold during training.

    :param      X: pd.DataFrame, predictor variables
    :param      y: pd.Series, target variable

    :return
    """

    # Create a list that will store the accuracies of each fold
    accuracy = []

    # Enter a loop to run K folds of cross-validation
    for fold in range(0, K):

        # Instantiate algorithm and scaler
        model = RandomForestRegressor()
        scaler = StandardScaler()

        # Create training and test samples
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=SPLIT,
random_state=42)

        # Scale X data, we scale the data because it helps the algorithm to converge
        # and helps the algorithm to not be greedy with large values
        scaler.fit(X_train)
        X_train = scaler.transform(X_train)
        X_test = scaler.transform(X_test)

        # Train model
        trained_model = model.fit(X_train, y_train)

        # Generate predictions on test sample
        y_pred = trained_model.predict(X_test)

        # Compute accuracy, using mean absolute error
        mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
        accuracy.append(mae)
        print(f"Fold {fold + 1}: MAE = {mae:.3f}")

    # Finish by computing the average MAE across all folds
    print(f"Average MAE: {(sum(accuracy) / len(accuracy)):.2f}")

# --- 4) MAIN FUNCTION
# Your algorithm code should contain modular code that can be run independently.
# You may want to include a final function that ties everything together, to allow
# the entire pipeline of loading the data and training the algorithm to be run all
# at once

# Execute training pipeline
def run():
    """
    This function executes the training pipeline of loading the prepared
    dataset from a CSV file and training the machine learning model

    :param

    :return
    """

    # Load the data first
    df = load_data()

    # Now split the data into predictors and target variables
    X, y = create_target_and_predictors(data=df)

    # Finally, train the machine learning model
    train_algorithm_with_cross_validation(X=X, y=y)
```