```
from futures3 import ProcessPoolExecutor, as_completed
from importlib import import_module
import os
import pandas as pd

from src.settings import REGISTRY, TEST_SIZE
from utils.fms import get_registry_dir_path, get_registry_outputdir_path, get_registry_codes


def all_registries_pipeline(verbose=False, read_only=False, is_test=False, test_size=TEST_SIZE):
    """
    Runs steps to extract aircraft registry data from all remote sources, transform the extracted data into a
    standardized format, and load the transformed data into tabular stores used to seed or update components of a data
    warehouse

    :param (bool) verbose: if 'True' print progress messages
    :param (bool) read_only: if 'True' skip the ETL process for each remote source and read the existing output instead
    :param (bool) is_test: if 'True' sample each registry data source for development purposes
    :param (int) test_size: number of observations to sample from registry data sources when 'test' argument is 'True'
    :return: wrangled world registry data;
        effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR]/[registry_code]/[file or directory] asset(s) for each remote source;
        effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_OUTPUT_DIR]/[registry_code]/[registry_code]
[REGISTRY.name] file for each remote source
    """

    # initialize storage for wrangled registry data
    registries = []

    # initialize and start parallel processing
    with ProcessPoolExecutor() as executor:
        if not read_only:
            # extract, transform, and load each remote registry data source
            futures = (executor.submit(
                single_registry_pipeline, registry_code, verbose, is_test, test_size
            ) for registry_code in get_registry_codes())
        else:
            # read already existing wrangled registry data
            futures = (executor.submit(
                read_wrangled_registry, registry_code, verbose, is_test
            ) for registry_code in get_registry_codes())

        for future in as_completed(futures):
            registry = future.result()

            if registry is not None:
                registries.append(registry)

    # compile world registry data
    registry = pd.concat(registries, ignore_index=True)

    # write world registry data
    path = os.path.join(get_registry_dir_path(is_test), "world" + REGISTRY["name"])
    registry.to_csv(path, index=False, line_terminator="\n")

    return registry


def single_registry_pipeline(registry_code, verbose=False, is_test=False, test_size=TEST_SIZE):
    """
    Runs steps to extract aircraft registry data from a specific remote source, transform the extracted data into a
    standardized format, and load the transformed data by integrating it into tabular stores used to seed or update
    components of a data warehouse

    :param (str) registry_code: registry-code of remote source to process
    :param (bool) verbose: if 'True' print progress messages
    :param (bool) is_test: if 'True' sample registry data source for development purposes
    :param (int) test_size: number of observations to sample from registry data source when 'test' argument is 'True'
    :return: remote source's wrangled registry data;
        effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR]/[registry_code]/[file or directory] asset(s);
        effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_OUTPUT_DIR]/[registry_code]/[registry_code][REGISTRY.name] file
    """
    # build path to class definition of specified registry source
    path = "registries." + registry_code.lower() + "." + registry_code.lower()
    # assign pseudonym for calling specified registry source's class
    registry = getattr(import_module(path), registry_code.upper())
    # initialize instance of specified registry source's class and assign wrangled registry data
    registry = registry(verbose=verbose, is_test=is_test, test_size=test_size).registry

    return registry


def read_wrangled_registry(registry_code, verbose=False, is_test=False):
    """
    Reads already existing wrangled aircraft registry data for a specific remote source

    :param (str) registry_code: registry-code of remote source to read wrangled output for
    :param (bool) verbose: if 'True' print progress messages
    :param (bool) is_test: if 'True' the session is in a testing-context
    :return: remote source's wrangled registry data
    """

    # read wrangled registry data
    filename = registry_code.lower() + REGISTRY["name"]
    path = os.path.join(get_registry_outputdir_path(is_test), registry_code.lower(), filename)

    try:
        if verbose:
            print("Progress: merging", registry_code, "into world registry data")

        registry = pd.read_csv(path, sep=",", memory_map=True, low_memory=False)
    except FileNotFoundError:
        print("Warning: wrangled registry for", registry_code, "does not exist and can't be merged into world registry")

        registry = None

    return registry
```