```
import pandas as pd
import re

from extractors.direct_downloads import extract_with_request, extract_with_headless_browser
from models.registry import Registry
from utils.helpers import clean_str, date_to_iso


class UKR(Registry):
    def __init__(self, **kwargs):
        """
        Child container for extracting, transforming, and loading registry data for Ukraine

        :param kwargs: see models.registry for details
        """

        super().__init__(**kwargs)

        # assign constant values for final registry
        self.source_url = "https://arcia.gov.ua/State-Civil-Aircraft-Register-of-Ukraine/"
        self.is_official = True
        self.source_name = "State Aviation Service of Ukraine"
        self.registry_code = "UKR"

        # assign registry input data source and filename
        self.registry_url = "https://arcia.gov.ua/register_of_aircraft.xls"
        self.registry_filename = "register_of_aircraft.xls"

        # assign registry input data extraction method
        self.extraction = self.extract_registry
        # assign updated-date scraping method for registry input data
        self.updated_scraper = self.scrape_registry_updated
        # assign registry input data reading method
        self.reader = self.read_registry
        # assign registry data wrangling method
        self.wrangler = self.wrangle_registry

        # extract, transform, and load registry data
        self.etl_registry()

    def extract_registry(self):
        """
        Extracts registry input data from remote source

        :return: effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR]/[self.country_code]/[self.registry_filename] file
        """

        # extract registry input data
        registry = extract_with_request(self.registry_url, self.registry_filename, registry_code=self.registry_code,
                                        is_test=self.is_test)

        # read registry input data
        self.read_registry(registry=registry)

    def scrape_registry_updated(self):
        """
        Scrapes and formats updated-date for registry input data from remote source

        :return: effect - modifies self.updated
        """

        soup = extract_with_headless_browser(self.source_url)
        date = None

        for section in soup.find_all("section"):
            date = section.find(text=re.compile("\d{1,2}.\d{1,2}.\d{4}"))

            if date is not None:
                date = date_to_iso(date, "%d.%m.%Y")
                break

        # assign updated-date for registry input data
        self.updated = date

    def read_registry(self, registry=None):
        """
        Reads registry input data

        :param (bytes) registry: stream of extracted registry input data
        :return: effect - modifies self.registry
        """

        if registry is None:
            # handle when registry input data has been extracted but not in current session
            registry = self.get_input_file_path(self.registry_filename)

        # read registry input data
        registry = pd.read_excel(registry, header=5, index_col=0)
        # prep registry input data
        registry.columns = [self.__format_colname(col) for col in registry.columns]

        # assign registry input data
        self.set_input_registry(registry)

    def wrangle_registry(self):
        """
        Transforms registry data

        :return: effect - modifies self.wrangled_records
        """

        # loop through raw registrations
        for row in self.registry.itertuples(index=False):
            # initialize storage for wrangled representation of registration in focus
            record = {
                "start_date": date_to_iso(row.Registration_certificate_issue_date, "%d.%m.%y"),
                "mark": row.Nationality_and_Registration_Mark,
                "model": row.Aircraft_Type,
                "serial_no": row.Serial_number,
                "year": row.Date_of_manufacture,
                "name": None,
                "is_owner": None,
                "is_operator": None,
            }

            # update list of transformed values to record registrant and accommodate multiple registrants
            self.__add_registrant_copies(row, record)

    def __add_registrant_copies(self, row, record):
        """
        Adds copies of a wrangled registration with updated registrant information to reflect all registrants

        :param (namedtuple) row: raw observation of a registration to extract registrants from
        :param (dict) record: wrangled representation of 'row' argument to duplicate
        :return: effect - modifies self.wrangled_records
        """

        # avoid modifying original wrangled representation of registration
        record_copy = record.copy()

        # parse multiple registered owners
        owners = row.Owners.split("/")

        # loop through registered owners
        for owner in owners:
            # avoid modifying original wrangled representation of registration
            record = record_copy.copy()

            # update registrant-related values
            record["name"] = clean_str(owner)
            record["is_owner"] = True

            # update list of transformed values to populate final registry
            self.wrangled_records.append(record)

        # check if registered operator value is "Does not apply"
        if row.Operator != "Не відповідає":
            # avoid modifying original wrangled representation of registration
            record = record_copy.copy()

            # update registrant-related values
            record["name"] = clean_str(row.Operator)
            record["is_operator"] = True

            # update list of transformed values to populate final registry
            self.wrangled_records.append(record)

    @staticmethod
    def __format_colname(colname):
        """
        Formats column name for namedtuple compatibility

        :param (str) colname: column name from registry input data to format
        :return: formatted copy of 'colname' argument suitable for namedtuple indexing
        """

        colname = colname.replace(" ", "_").replace("_", "")

        return colname
```