```
import camelot
import pandas as pd
import re

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


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

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

        super().__init__(**kwargs)

        # assign constant values for final registry
        self.source_url = "https://caa.gov.mv/rules-and-regulations/maldives-civil-aviation-act"
        self.is_official = True
        self.source_name = "Maldives Civil Aviation Authority"
        self.registry_code = "MDV"

        # assign registry input data source and filename
        self.registry_url = "https://spreadsheets.google.com/e/aviation.gov.mv/spreadsheet/pub?
hl=en_US&single=true&gid=0&output=pdf"
        self.registry_filename = "DLT Registries.pdf"

        # assign registry input data extraction method
        self.extractor = self.extract_registry
        # 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
        extract_with_request(self.registry_url, self.registry_filename, registry_code=self.registry_code,
                             is_test=self.is_test)

    def read_registry(self):
        """
        Reads registry input data

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

        # read registry input data
        registry = camelot.read_pdf(self.get_inputfile_path(self.registry_filename), pages="1-end")
        # prep registry input data
        registry = self.__format_registry(registry)
        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 = {
                "date": row.Registration,
                "year": hexadecimal_to_decimal(row.DLT_MSN_Index.replace(" ", "")),
                "model": row.dlt_type,
                "serial_no": row.DLT_SN
            }

            self.wrangled_records.append(record)

    def __format_registry(self, registry):
        """
        Cleans and merges tables across multiple pdf pages

        :param (camelot.core.TableList) registry: tables of registrations from registry pdf to process
        :return: cleaned and merged transformation of 'registry' argument
        """

        # initialize storage for tables on multiple pages
        tables = []
        # initialize storage for column names
        header = None
        # initialize storage for confirmation notes
        notes = []

        # loop through pdf pages
        for table in registry:
            # assign table on the page in focus
            table = table.df

            if header is None:
                # assign column names once using the table on the first page
                header = table.iloc[0, 0].split("\n")

            # drop header row from each table
            table.drop(0, inplace=True)
            # assign indices of rows with confirmation notes
            indices = table[table[0].str.contains("Confirmed")].index
            # update list of confirmation notes
            notes.extend(table.iloc[i, 0] for i in indices)
            # drop rows with confirmation notes
            table.drop(indices, inplace=True)

            # loop through rows where all values are concatenated into the first column
            for i in table[table.iloc[:, 1:].apply(lambda x: x.str.len()).sum(axis=1) == 0].index.tolist():
                # extend values into correct columns
                table.iloc[i] = table.iloc[i, 0].split("\n")

            tables.append(table)

        # merge tables across multiple pages together
        registry = pd.concat(tables, ignore_index=True)
        registry.columns = header

        # assign updated-date for registry input data
        self.__set_updated(notes)

        return registry

    def __set_updated(self, notes):
        """
        Parses and formats updated-date for registry input data from confirmation notes

        :param (list) notes: confirmation notes from registry input data
        :return: effect - modifies self.updated
        """

        # initialize storage for dates parsed from confirmation notes
        dates = []

        # loop through confirmation notes
        for note in notes:
            # parse date of note in focus
            note = note.replace("\n", " ")
            date = re.search("(\d{1,2} \w{3,} \d{4})", note).group()
            # format date of note in focus
            date = date_to_iso(date, "%d %B %Y")

            dates.append(date)

        # use the oldest confirmation note to assign updated-date for registry input data
        self.updated = min(dates)

    @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
```