```
from datetime import datetime
import dateutil.parser as dparser
import pandas as pd
from PyPDF2 import PdfFileReader
import re


def handle_cli_verbosity(argv, min_num):
    """
    Validates the 'verbose' command line argument passed when a registry data pipeline script is run

    :param (list) argv: command line arguments
    :param (int) min_num: minimum number of command line arguments to be provided for the given data pipeline script
    :return: 'True' if the 'verbose' argument was properly passed and progress messages should be printed, else 'False'
    """

    if len(argv) > min_num and argv[min_num].lower() == "verbose":
        # handle when argument is appropriately provided
        verbose = True
    elif len(argv) > min_num and argv[min_num].lower() != "verbose":
        # handle when an invalid argument is provided
        print("Warning: unrecognized value for argument #" + str(min_num + 1) + " (verbose), verbose is set to 'False'")

        verbose = False
    else:
        # handle when argument is not provided
        verbose = False

    return verbose


def ld_to_dl(l_of_d):
    """
    Transforms a list of dictionaries to a dictionary of lists by merging key-value pairs

    :param (list) l_of_d: list of dictionaries to transform
    :return: dictionary of lists transformation
    """

    d_of_l = {key: [dic[key] for dic in l_of_d] for key in l_of_d[0]}

    return d_of_l


def clean_str(string, nullable=True):
    """
    Cleans string by removing extra whitespace and new-line characters

    :param (str) string: string value to clean
    :param (bool) nullable: if 'True' return empty string as 'None', if 'False' return empty string
    :return: cleaned copy of 'string' argument
    """

    if pd.isnull(string):
        return None if nullable else ""

    # remove extra whitespace and new-line characters
    string = "".join(" ".join(str(string).split()).split("\n")[0]

    if string == "" or string == "N/A":
        # handle when string is empty or empty-like characters
        return None if nullable else ""

    return string


def str_to_int(string):
    """
    Converts string value to integer equivalent

    :param (str or int) string: string value to convert to integer value
    :return: integer equivalent of 'string' argument
    """

    if pd.isnull(string):
        return None
    elif isinstance(string, (int, float, complex)):
        # handle when string is already a numeric value
        return string

    string = clean_str(string)

    if string is not None:
        string = int(string)

    return string


def num_to_str(num, integer=False):
    """
    Converts numeric value to string equivalent

    :param (numeric or str) num: numeric value to convert to string value
    :param (bool) integer: if 'True' cast 'num' argument as an integer value with no rounding
    :return: string equivalent of 'num' argument
    """

    if pd.isnull(num):
        return None
    elif isinstance(num, str):
        # handle when num is already a string value
        num = clean_str(num)
    elif integer:
        num = int(num)

    num = clean_str(str(num))

    return num


def squeeze_chr(character, string):
    """
    Reduces consecutive repeating instances of a specific character in a string to a single instance

    :param (str) character: target character to reduce instances of
    :param (str) string: string with instances of 'character' argument to modify
    :return: copy of 'string' argument with reduced instances of 'character' argument
    """

    while character * 2 in string:
        string = string.replace(character * 2, character)

    return string


def drop_parentheticals(string, nullable=True):
    """
    Removes parenthetical text and open/close parenthesis characters from a string

    :param (str) string: string to remove parenthetical text from
    :param (bool) nullable: if 'True' assign empty strings as 'None', if 'False' assign empty string
    :return: formatted copy of 'string' value with parenthetical text removed
    """

    while "(" in string:
        try:
            # drop parenthetical text
            string = string.replace("(" + re.search(r"\(([^)]+)", string).group(1) + ")", "")
        except AttributeError:
            # handle corner-cases like mismatching parentheses
            string = string.replace("(", "").replace(")", "")

    # remove trailing whitespaces and redundant whitespaces introduced by removing parentheses
    string = clean_str(squeeze_chr(" ", string), nullable)

    return string


def date_to_iso(date, form):
    """
    Converts date to ISO format

    :param (str) date: date to convert
    :param (str) form: format that represents the structure of the 'date' argument
    :return: copy of 'date' argument in ISO format
    """

    date = clean_str(date)

    if date is None:
        return None

    date = datetime.strptime(date, form).isoformat()

    return date


def date_from_pdf(stream, re_format, date_format, replace=None):
    """
    Parses and formats the date from a pdf's header or footer sections

    :param (io.BytesIO or str) stream: file-like object or file path of the pdf to parse the date of interest from
    :param (str) re_format: regex pattern used to search and isolate the date of interest
    :param (str) date_format: expected format of the date isolated by the 'regex' argument
    :param (tuple or None) replace: arguments to pass to str.replace() to facilitate regex search
    :return: pdf's header or footer date in ISO format
    """

    if isinstance(stream, str):
        # handle when stream argument is a path to pdf file
        stream = open(stream, "rb")

    pdf = PdfFileReader(stream)
    text = pdf.getPage(0).extractText()

    if replace is not None:
        text = text.replace(replace[0], replace[1])

    date = re.search(re_format, text).group()
    date = date_to_iso(date, date_format)

    return date


def parse_date(string):
    """
    Parses the date from a string and converts it to ISO format

    :param string: text to parse date from
    :return: ISO formatted date contained in 'string' argument
    """

    date = dparser.parse(string, fuzzy=True).isoformat()

    return date


def build_address(template, components):
    """
    Cleans and formats human-readable address from separated components

    :param (str) template: country-specific common address format to use for interpolation
    :param (list) components: address components to be interpolated into 'template' argument
    :return: cleaned and formatted address
    """

    components = [clean_str(x, nullable=False) for x in components]

    if len(pd.unique(components)) <= 1:
        # handle when all address components are equal, such as all None or all empty strings
        return None

    # organize address components into common address format
    address = template.format(*components)
    # format commas and remove empty strings
    address = " ".join(address.split()).replace(", ", ",").replace(" ,", ",")
    # reduce consecutive repeating commas
    address = squeeze_chr(",", address)

    if address[0] == ",":
        # drop leading comma
        address = address[1:]

    if address[-1] == ",":
        # drop trailing comma
        address = address[:-1]

    # pad commas with subsequent whitespace
    address = address.replace(",", ", ")

    return address


def sanitize_str_cols(df, nullable=True):
    """
    Cleans and formats string columns to facilitate SQL table integration

    :param (pandas.DataFrame) df: dataframe with string columns to sanitize
    :param (bool) nullable: if 'True' assign empty strings as 'None', if 'False' assign empty string
    :return: cleaned and formatted copy of 'df' argument
    """

    # loop through string columns
    for col in df.columns[df.applymap(lambda x: isinstance(x, str)).any()]:
        df[col] = df[col].apply(lambda x: clean_str(x, False))
        # convert ", ', and ` characters to apostrophe
        df[col] = df[col].apply(lambda x: x.replace("\'", "'").replace("``", "'").replace("`", "'"))
        df[col] = df[col].apply(lambda x: clean_str(x, nullable))

    return df


def hexadecimal_to_decimal(hexadecimal):
    """
    Converts hexadecimal Mode-S Transponder Code value to decimal equivalent

    :param (str) hexadecimal: hexadecimal Mode-S Transponder Code to convert
    :return: copy of 'hexadecimal' argument in base-16 integer
    """

    hexadecimal = clean_str(hexadecimal)

    return None if hexadecimal is None else int(hexadecimal, 16)


def binary_to_decimal(binary):
    """
    Converts binary Mode-S Transponder Code value to decimal equivalent

    See https://www.geeksforgeeks.org/binary-decimal-vice-versa-python/ for reference

    :param (int) binary: binary Mode-S Transponder Code to convert
    :return: copy of 'binary' argument in base-16 integer
    """

    binary = str_to_int(binary)
    decimal = None

    if not pd.isnull(binary):
        decimal = 0
        i = 0

        while binary != 0:
            decimal = decimal + (binary % 10) * pow(2, i)
            binary = binary // 10
            i += 1

    return decimal
```