Task Report: Assignment 1

This is my report outlining the steps taken to develop a Python code for
normalizing and processing dates from a given text file. The code uses regular
expressions to identify and normalize dates into a standard format (`YYYY-MM-DD`),
and then applies a 40-day offset to each normalized date. The final output includes
the original normalized date, the offset date, and the string of the offset date.

The following are the regular expressions I used to capture various date formats:

1. Pattern 1: `\b(\d{1,2})\s*(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|
January|February|March|April|May|June|July|August|September|October|November|
December)[a-z]*\.?,?\s*(\d{4})`
It matches dates in the format "day month year" (e.g., "15 Jan 2023").

2. Pattern 2: `\b(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|
March|April|May|June|July|August|September|October|November|December)[a-z]*\.?,?\
s*(\d{1,2})?,?\s*(\d{4})`
It matches dates in the format "month day year" (e.g., "Jan 15, 2023").

3. Pattern 3: `(\d{1,2})\s*(Jan|Feb|Mar|Marc|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
[a-z]*\.?,?\s*(\d{2,4})`
It matches dates with a two-digit year (e.g., "15 Jan 23").

4. Pattern 4: `(Jan|Feb|Mar|Marc|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|
February|March|April|May|June|July|August|September|October|November|December)[a-
z]*\.?,?\s*(\d{4})`
It matches dates in the format "month year" (e.g., "Jan 2023").

5. Pattern 5: `(Jan|Feb|Mar|Marc|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\.?,?\
s*(\d{1,2})?,?\s*(\d{2,4})`
It matches dates with optional day and two-digit year (e.g., "Jan 15, 23").

6. Pattern 6: `(January|February|March|April|May|June|July|August|September|
October|November|December)\s*\.?,?\s*(\d{1,2})?,?\s*(\d{2,4})`
It matches full month names with optional day and two-digit year.

7. Pattern 7: `(Jan|Feb|Mar|Marc|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|
February|March|April|May|June|July|August|September|October|November|December)[a-
z]*\.?,?\s*(\d{2,4})`
It matches month with two-digit year (e.g., "Jan 23").

8. Pattern 8: `(\d{1,2})[/-](\d{1,2})[/-](\d{2,4})`
It matches dates in the format "day/month/year" or "day-month-year".

9. Pattern 9: `(\d{1,2})[/-](\d{4})`
It matches dates in the format "day/year".

10. Pattern 10: `(\d{4})`
It matches standalone four-digit years.

The following are the cleaning steps and assumptions I made:

1. Removing Phone Numbers: The input text was cleaned by removing any occurrences
of phone numbers in the format "XXX-XXX-XXXX" using the regular expression `\b\
d{3}-\d{3}-\d{4}\b`.

2. Two-Digit Year Handling: For dates with two-digit years, it was assumed that
years less than 100 belong to the 20th century (i.e., years 00-99 are treated as

1900-1999).

3. Missing Day and Month Handling: When the day of the month was not specified, it was assumed to be the first day of the month. When the month was not specified it was assumed to be January.

4. Month Abbreviations: The code uses both abbreviated and full month names for parsing and formatting dates. "Marc" as an abbreviation for March was additionally included as it was detected in an entry.

5. In case of no date in entry: If an entry does not contain a date, it will be ignored and not included in the output file.

The program reads a text file line by line, where each line contains a line number and a date string separated by a tab. The date normalization function matches each date string against the predefined regular expression patterns to extract the date components (day, month, year). Once a date is normalized to the `YYYY-MM-DD` format, a 40-day offset is applied using Python's `datetime` and `timedelta` classes. The original normalized date, the offset date in `YYYY-MM-DD` format, and the offset date in a string format (month day, year) are written to an output file.