FIT5196 Task 1 in Assessment 1

Student Name: Anh Huy Phung

Student ID: 34140298

Student Name: Wei Yu Su

Student ID: 33642605

Date: 19-04-2024

Environment: Python 3.10

Libraries used:

- re (for regular expression, installed and imported)
- pandas (for data manipulation)
- json (for loading/dumping the files)

Table of Contents

---

1. Introduction

This assessment regards extracting trademark data from xml text files with allowance of using 3 libraries: re, pandas, json. The dataset is provided on shared google drive, which can be found in this link.

2. Importing Libraries

The packages to be used in this assessment are imported in the following. They are used to fulfill the following tasks:

- **re:** is main library in this task, to define and use regular expressions, capture information in xml tagname and remove person's title
- **pandas:** to convert date format
- **json:** to load/dump information json from/to files

```
import re
import pandas as pd
import json
```

3. Examining Raw Data

3.1 First of all, these are attribute descriptions we need to transform:

- **rf_id:** Unique ID which is a combination of reel-no and frame-no
- **last-update-date:** The date that the assignment record was last modified
- **conveyance-text:** Textual description of the interest conveyed or transaction recorded
- **correspondent-party:** Name of a person or organisation to whom correspondence related to the assignment
- **assignors-info:**
  - **party-name:** Person or organisation name that conveys an interest or transaction, remove the title of party is person
  - **date-acknowledged:** Supporting legal documentation was acknowledged
  - **execution-date:** Supporting legal documentation was executed
  - **country:** Party's country. Put 'USA' or 'UK' if the country name is a variant of USA or UK. Fill 'NA' if the country can't be defined
  - **legal-entity-text:** The party's legal entity
- **assignees-info:**
  - **party-name:** Person or organisation name that receiving an interest or transaction, remove the title of the party is person
  - **country:** Party's country. Put 'USA' or 'UK' if the country name is a variant of USA or UK. Fill 'NA' if the country can't be defined
  - **legal-entity-text:** The party's legal entity
- **property-count:** Number of properties for trademark assignment entry

3.2 Having examined the file content, the following observations were made:

Here is the demonstration for one assignment:

```
<assignment-entry>
      <assignment>
          <reel-no>228</reel-no>
          <frame-no>0069</frame-no>
          <last-update-date>19920130</last-update-date>
          <purge-indicator>N</purge-indicator>
          <date-recorded>19721103</date-recorded>
          <page-count>2</page-count>
          <correspondent>
              <person-or-organization-name>CLUETT, PEABODY &amp;amp; CO., INC.</person-or-organization-name>
              <address-1>530 FIFTH AVE.</address-1>
              <address-2>NEW YORK, NY 10036</address-2>
          </correspondent>
          <conveyance-text>ASSIGNS THE ENTIRE INTEREST AND THE GOODWILL</conveyance-text>
      </assignment>
      <assignors>
          <assignor>
              <person-or-organization-name>VAN RAALTE COMPANY, INC.</person-or-organization-name>
              <address-2>417 FIFTH AVE.</address-2>
              <city>NEW YORK</city>
              <state>NEW YORK</state>
              <postcode>10016</postcode>
              <date-acknowledged>19721101</date-acknowledged>
              <legal-entity-text>CORPORATION</legal-entity-text>
              <nationality>NEW YORK</nationality>
          </assignor>
      </assignors>
      <assignees>
          <assignee>
              <person-or-organization-name>CLUETT, PEABODY &amp;amp; CO., INC.</person-or-organization-name>
              <address-2>530 FIFTH AVE.</address-2>
              <city>NEW YORK</city>
              <state>NEW YORK</state>
              <postcode>10036</postcode>
              <legal-entity-text>CORPORATION</legal-entity-text>
              <nationality>NEW YORK</nationality>
          </assignee>
      </assignees>
      <properties>
          <property>
              <serial-no>72238520</serial-no>
              <registration-no>820810</registration-no>
          </property>
      </properties>
    </assignment-entry>
```

From above definition and demonstration, we need to classify further these attributes:

- **party-name:**
  - **Personal or organization classification:** In this case we use information tagname `legal-entity-text` or any `organizational shortcut characters` (eg: llc, inc, etc) to classify. If there no match in these organization's signals, we classify it as personal's name and remove title.
  - **Remove titles:** Personal's titles lie mostly in 2 positions: beginning (eg: 'MR. ') and end (eg: ', MR.') of the name, hence we need to choose regex pattern for each case to remove these.

```
<assignees>
    <assignee>
        <person-or-organization-name>AUSTIN, GREG, MR.</person-or-organization-name>
        <address-1>1677 WASHINGTON AVENUE S.</address-1>
        <city>EDEN PRAIRIE</city>
        <state>MINNESOTA</state>
        <postcode>55344</postcode>
        <legal-entity-text>INDIVIDUAL</legal-entity-text>
        <nationality>UNITED STATES</nationality>
    </assignee>
<assignors>
    <assignor>
        <person-or-organization-name>MR. GASKET COMPANY</person-or-organization-name>
        <execution-date>19930430</execution-date>
        <legal-entity-text>CORPORATION</legal-entity-text>
        <nationality>OHIO</nationality>
    </assignor>
</assignors>
```

- **country:**
  - There are attributes to consider regarding to the country info: '**nationality**', '**country-name**', '**state**', and '**city**'. Out of 4 attributes, '**nationality**' and '**country-name**' have the highest correlation in terms of country's information but only **'nationality'** stays recorded for assignors and assignees while '**country-name**' appears randomly
  - After discovering the data, there some mismatch about these and we have to put it into priorities. For example, **'nationality'** might be different from **'country-name'**. In this example below country-name is: *ONTARIO* but actually this is one province of *CANADA*, which means **'nationality'** is correct this time.

```
    </assignors>
    <assignees>
        <assignee>
            <person-or-organization-name>NUTRIMIX LABORATORIES, INC.</person-or-organization-name>
            <address-1>2487 KALADAR AVENUE</address-1>
            <address-2>SUITE 360E</address-2>
            <city>OTTOWA</city>
            <country-name>ONTARIO</country-name>
            <postcode>K1V8B9</postcode>
            <legal-entity-text>CORPORATION</legal-entity-text>
            <nationality>CANADA</nationality>
```

  - To tackle such issue, we follow 2 steps:
    1. The rank above 4 tagnames from highest to lowest priority for **assignor** and **assignee**
    2. In each prioritized tagname, we classify country by :
       - **Main countries (USA, UK and CANADA):** In each of which we prepare its related-information (common-called name, states or provinces). If it matches one of these information, classify it into corresponding the country.
       - **Other countries:** Take the data directly if it is not in the main countries' information and not 'NA'
       - Continue to examine the next tagname if current tagname information can't defined ('NA', 'NOT PROVIDED', etc)

- **assignors-info:**
  - Country-related tagnames towards assignors is mostly **'nationality'** and some cases there will be missed some tagnames like **'country-name'**, **'state'**, **'city'** to validate country information.

- This example show an example that we have full tagnames about assignors

```
<assignors>
    <assignor>
        <person-or-organization-name>VAN RAALTE COMPANY, INC.</person-or-organization-name>
        <address-2>417 FIFTH AVE.</address-2>
        <city>NEW YORK</city>
        <state>NEW YORK</state>
        <postcode>10016</postcode>
        <date-acknowledged>19721101</date-acknowledged>
        <legal-entity-text>CORPORATION</legal-entity-text>
        <nationality>NEW YORK</nationality>
    </assignor>
</assignors>
<assignees>
    <assignee>
        <person-or-organization-name>CLUETT, PEABODY &amp;amp; CO., INC.</person-or-organization-name>
        <address-2>530 FIFTH AVE.</address-2>
        <city>NEW YORK</city>
        <state>NEW YORK</state>
        <postcode>10036</postcode>
        <legal-entity-text>CORPORATION</legal-entity-text>
        <nationality>NEW YORK</nationality>
```

- This shows an example some tagnames are missed, only `nationality` tagname exists

```
<assignors>
    <assignor>
        <person-or-organization-name>SOTAN PTY, LTD.</person-or-organization-name>
        <execution-date>20010913</execution-date>
        <legal-entity-text>CORPORATION</legal-entity-text>
        <nationality>AUSTRALIA</nationality>
    </assignor>
</assignors>
<assignees>
    <assignee>
        <person-or-organization-name>COOGI IP PTY, LTD.</person-or-organization-name>
        <address-1>452 JOHNSTON STREET</address-1>
        <city>ABBOTSFORD, VICTORIA 3067</city>
        <country-name>AUSTRALIA</country-name>
        <legal-entity-text>CORPORATION</legal-entity-text>
        <nationality>AUSTRALIA</nationality>
    </assignee>
```

  - Hence, we allocate priority to validate country by following tagnames: **'nationality'**, **'country-name'**, **'state'**, **'city'**

- **assignees-info:**:
  - Country-related information towards assignees is still 4 attributes **'country-name'**, **'state'**, **'city'**, and **'nationality'** but in some cases there will be missed tagnames like **country-name**, and **'city'** to validate country information. For Assignee's cases, **'nationality'** may be not the address of the assignees as a registered country can be different from nationality.
    - Below show us an example of assignee has nationality **INDIA** but his country location now is **US**.

```
<assignees>
    <assignee>
        <person-or-organization-name>RAMNARAYAN, KAL</person-or-organization-name>
        <address-1>12279 OAKVIEW WAY</address-1>
        <city>SAN DIEGO</city>
        <state>CALIFORNIA</state>
        <postcode>92128</postcode>
        <legal-entity-text>INDIVIDUAL</legal-entity-text>
        <nationality>INDIA</nationality>
    </assignee>
```

- This is another example that nationality is **SINGAPORE** but the current country is **US**

```
<assignees>
    <assignee>
        <person-or-organization-name>WAVE LIFE SCIENCES LTD.</person-or-organization-name>
        <address-1>8 CROSS STREET</address-1>
        <address-2>#10-00 PWC BUILDING</address-2>
        <city>SINGAPORE</city>
        <country-name>UNITED STATES</country-name>
        <postcode>048424</postcode>
        <legal-entity-text>PUBLIC COMPANY</legal-entity-text>
        <nationality>SINGAPORE</nationality>
```

  - Hence, we allocate priority to validate country by following tagnames: **'country-name'**, **'state'**, **'nationality'**, **'city'**

- **property-count:**
  - There are some identical duplicate properties regarding to property, so we eliminate those serial numbers already existed when counting.

```
<properties>
    <property>
        <serial-no>75418582</serial-no>
        <registration-no>2232041</registration-no>
    </property>
    <property>
        <serial-no>75418582</serial-no>
        <registration-no>2232041</registration-no>
    </property>
```

4. Generate Extraction Functions

In this session, we will explain our methodology to approach this assignment. The general concept is we will extract information from each assignment, then continue to validate the information of the entities described the section 3, validate these data and save all of entities's information under the corresponding assignment's unique id (**rf_id**). Finally, store all of the data in JSON format.

4.1. Tagname information

In this section, we will demonstrate our functions using regex to extract data from tagnames. For simplicity, we will use one piece of assignment's information named 'raw_info' to show as an example and after we will do all data extraction and comparison in section 5.

```
# This is a piece of assignment including all of entities's information
# Each of assignment information will lie between two <assignment-entry> tagnames
raw_info ='''
<assignment-entry>
      <assignment>
          <reel-no>4656</reel-no>
          <frame-no>0341</frame-no>
          <last-update-date>20111108</last-update-date>
          <purge-indicator>N</purge-indicator>
          <date-recorded>20111107</date-recorded>
          <page-count>7</page-count>
          <correspondent>
              <person-or-organization-name>JENNIFER E. LACROIX</person-or-organization-name>
              <address-1>P. O. BOX 64807</address-1>
              <address-2>CHICAGO, IL 60664-0807</address-2>
          </correspondent>
          <conveyance-text>ASSIGNS THE ENTIRE INTEREST</conveyance-text>
      </assignment>
      <assignors>
```

```
            <assignor>
                <person-or-organization-name>FRITZ COMPANY, INC.</person-or-organization-name>
                <execution-date>20111107</execution-date>
                <legal-entity-text>CORPORATION</legal-entity-text>
                <nationality>MINNESOTA</nationality>
            </assignor>
            <assignor>
                <person-or-organization-name>FRITZ SALES WORLDWIDE, INC.</person-or-organization-name>
                <execution-date>20111107</execution-date>
                <legal-entity-text>CORPORATION</legal-entity-text>
                <nationality>MINNESOTA</nationality>
            </assignor>
        </assignors>
        <assignees>
            <assignee>
                <person-or-organization-name>TRUDEAU FOODS, LLC</person-or-organization-name>
                <address-1>25 WEST CLIFF ROAD</address-1>
                <address-2>SUITE 115</address-2>
                <city>BURNSVILLE</city>
                <state>MINNESOTA</state>
                <postcode>55337</postcode>
                <legal-entity-text>LIMITED LIABILITY COMPANY</legal-entity-text>
                <nationality>DELAWARE</nationality>
            </assignee>
        </assignees>
        <properties>
            <property>
                <serial-no>74291185</serial-no>
                <registration-no>1757091</registration-no>
            </property>
            <property>
                <serial-no>78282038</serial-no>
                <registration-no>2912935</registration-no>
            </property>
        </properties>
      </assignment-entry>
'''
```

Here are all functions used to extract information under tagnames:

1. **tagname_pattern:** deliveries tagname's regex pattern to extract information under such tagname
2. **check_empty:** validate helps us to valid an input list is empty or not and return 'NA' if it is empty
3. **convert_character:** convert special letters into desired ones, in this case, we transform **&** into **&**
4. **tagname_info:** Finally extract information under input tagname from a text, and we can decide whether collected data is cleaned or not by parameter get_detail_info with default is True.

Below are all the mentioned functions and example to examine it

```
def tagname_pattern(tag_name):
  """
  Returns the pattern for input tagname
  """
  return rf'<{tag_name}>(.*?)</{tag_name}>'

def check_empty(input_value):
  """
  Checks of input list value is empty or not, return 'NA' if the list is emtpy
  """
  if input_value:
    first_val = input_value[0]
    return first_val.strip()
  else:
    return 'NA'
def convert_character(party_name):
  """
  Replace special character '&amp;amp;' into '&'
  """
  cleaned_name = party_name
  if '&amp;amp;' in cleaned_name:
    cleaned_name = cleaned_name.replace('&amp;amp;', '&')
  return cleaned_name
def tagname_info(tag_name, text_info, get_detail_info = True):
  """
  This function gets all information within the tagname from a xml-formated text, decided by get_detail_info parameter:
```

```
  - get_detail_info is True then get all the within detail and transform it via check_empty function
  - get_detail_info is False then just return all the pattern-matches in text_info
  """
  pattern = tagname_pattern(tag_name)
  matches = re.findall(pattern, text_info, re.DOTALL)
  if get_detail_info:
    apply_functions = lambda x: convert_character(check_empty(x))
    return apply_functions(matches).strip()
  else:
    return matches
```

```
# This code extract all raw information under assignment-entry tagname without transformation
assignment_info = tagname_info('assignment-entry', raw_info, False)
print(assignment_info)
# This code extract all raw information under assignment-entry tagname with transformation (check empty, convert specia
reel_info = tagname_info('reel-no', raw_info, True)
print(reel_info)
```

4.2. Single attributes

Below are functions help us to extract information from **single attribute** in assignment (rf_id, last-update-date, conveyance_text and property_count)

1. **rf_id_converted:** Use to extract unique ID that is a combination of 2 tagnames reel-no and frame-no
2. **date_converted:** Extract the date that the assignment records under a specific tagname, then convert it into `yyyy-mm-dd` format in digits.
3. **conveyance-text:** Use above tagname_info to extract information under such tagname
4. **property_converted:** Extract all serial under property tagname, eliminates the duplicated ones and returns total properties

```
def rf_id_converted(reel_tag_name, frame_tag_name, text_info):
    """
    This function returns rf_id by combining reel_no and frame_no
    """
    reel_no = tagname_info(reel_tag_name, text_info, True)
    frame_no = tagname_info(frame_tag_name, text_info, True)
    return reel_no + '' + frame_no

def date_converted(date_tag_name, text_info):
    """
    This function converts date under specific tagname in a text string with format yyyy-mm-dd in digits
    """
    try:
      date_input = tagname_info(date_tag_name, text_info, True)
      if date_input == 'NA':
        return 'NA'
      new_date = pd.to_datetime(date_input, format = '%Y%m%d').strftime('%Y-%m-%d')
      return new_date
    except ValueError:
      print('Invalid date input or input not exist, try again')

def property_converted(text_info):
    """
    This function extracts all serial under property tag name, eliminates the duplicated ones and returns total properti
    """
    list_property = []

    split_data = tagname_info('property', text_info, False)
    for element in split_data:
      pattern_serial  = r'<serial-no>(.*?)</serial-no>'
      serial = re.findall(pattern_serial, element, re.DOTALL)
      if check_empty(serial) == 'NA':
        continue
      else:
        serial = check_empty(serial)
        if serial not in list_property:
          list_property.append(serial)
    return str(len(list_property))
```

```
# This code extract all raw information under assignment-entry tagname without transformation
assignment_info = rf_id_converted('reel-no','frame-no', raw_info)
```

```
print('rf_id: ', assignment_info)

# This code extract all raw information under assignment-entry tagname with transformation (check empty, convert specia
last_update_date = date_converted('last-update-date', raw_info)
print('last_update_date: ',last_update_date)

# This code also extract all raw information under conveyance-text tagname with transformation (check empty, convert sp
conveyance_text = tagname_info('conveyance-text', raw_info, True)
print('conveyance_text: ', conveyance_text)

# We can extract correspondent_party information under correspondent_party tagname with transformation (check empty, co
# Although we have many person-or-organization-name tagnames but the tagname_info function catch the first one, which i
correspondent_party = tagname_info('person-or-organization-name', raw_info, True)
print('correspondent_party: ', correspondent_party)

# This code extract all raw information under assignment-entry tagname without transformation
property_information = property_converted(raw_info)
print('property-count: ', property_information)
```

4.3. Complex attributes

Regarding to country, we need to create a function to categorize countries into 3 parts: big, undefined and other countries. After that, we will have another function to pick out the most suitable country's information from the input prioritized tagname list.

```
def check_country(input_country):
    """
    This function checks of input COUNTRY parameter is from which country:
      Big countries (USA, UK, CANADA) will be defined by their common-called name and provinces/states

      Return other countries if the input is defined
      Return False if the country can't be defined ('NOT PROVIDED','UNKNOWN','NA','')
    """
    if input_country == []:
      return 'NA'
    input_country = input_country.upper()

    # This US list has 2 parts, first one is states and second is common-called / old US name (eg: AMERICA)
    # We put it as 2 parts for easier investigation country's names
    us_list = [[ 'ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA', 'COLORADO', 'CONNECTICUT', 'DELAWARE', 'FLOR

    # This UK has all states and common-called / old UK name
    uk_list = ["ENGLAND", "SCOTLAND", "WALES", "NORTHERN IRELAND", 'UNITED KINGDOM', 'GREAT BRITAIN']

    # This CANADA has all states and common-called / old CANADA name
    canada_list = ['ALBERTA', 'BRITISH COLUMBIA', 'MANITOBA', 'NEW BRUNSWICK', 'NEWFOUNDLAND AND LABRADOR', 'NOVA SCOT
    if input_country in us_list[0] or input_country in us_list[1]:
      return 'USA'
    elif input_country in uk_list:
      return 'UK'
    elif input_country in canada_list:
      return 'CANADA'
    elif input_country in ['NOT PROVIDED','UNKNOWN','NA','']:
      return False
    else:
      return input_country.strip()

def country_converted(tag_name_list, text_info):
    """
    This function returns country from text via input list of prioritized tag name
      For each tag names in the list, we extract relative information from it and use check_country function to validat
      Continue to search if the current country's information can't be defined (False)
    """
    for tag_name in tag_name_list:
      pattern = tagname_pattern(tag_name)

      country_info = tagname_info(f'{tag_name}', text_info, True)

      # Use check_country function to validate country's information
      country_info = check_country(country_info)

      # Continue to search for country information in following tag names in the list if current one is undefined
      if country_info is False:
        continue
```

```
        else:
          return country_info
    # return NA if all the tagnames dont have the country information
    return 'NA'
```

```
"""
Below are the test codes for extracting country's information in assignor and assignee
Notice that one assignemnt can have many assignors and assignees
"""
# This code tests the information of assignor from prioritized tag name list: ['nationality', 'country-name', 'state',
assignor_info = tagname_info('assignor', raw_info, False)
for i, element in enumerate(assignor_info, start=1):
    country = country_converted(['nationality', 'country-name', 'state', 'city'], element)
    print(f"assignor{i}'s country:", country)

# This code tests the information of assignor from prioritized tag name list: ['country-name', 'state', 'nationality',
assignees_info = tagname_info('assignee', raw_info, False)
for i, element in enumerate(assignees_info, start=1):
  country =  country_converted(['country-name', 'state', 'nationality','city'], element)
print(f"assignee{i}'s country:", country)
```

The below functions also help us to extract the **party_name** and **entity_value** of each entity. Regarding the party_name classification, we will base the entity's value on classifying whether that entity is an organization or an individual. If an entity's value can't defined then we have to base it on organizational patterns in the name itself. Below are patterns that define individual patterns, remove titles and classify whether it's organization or not, all of these will be used in such functions.

```
# Individual pattern in the ENTITY VALUE
indi_pattern = r'\b(?:individual)\b'

# Individual title pattern in the name of party,
pattern_titles_comma = r',\s*(?:Mr\.|Mrs\.|Miss\.|Ms\.|Mx\.|Sir\.|Dame\.|Dr\.|Cllr\.|Lady\.|Lord\.)'
pattern_titles = r'(?:Mr\.|Mrs\.|Miss\.|Ms\.|Mx\.|Sir\.|Dame\.|Dr\.|Cllr\.|Lady\.|Lord\.)'

# Organization pattern in the name of party
org_pattern = r'\b(?:ltd|llp|llc|BV|B\.V\.|company|corporation|cor|bank|banking|inc|limited|sas|GMBH|S\.R\.L|SOCIÉTÉ)\b
```

```
def remove_person_titles(text_input):
    """
    This function replaces title at beginning and end of name
    Use for replace titles in the individual name
    """

    pattern_titles_comma = r',\s*(?:Mr\.|Mrs\.|Miss\.|Ms\.|Mx\.|Sir\.|Dame\.|Dr\.|Cllr\.|Lady\.|Lord\.)'
    pattern_titles = r'(?:Mr\.|Mrs\.|Miss\.|Ms\.|Mx\.|Sir\.|Dame\.|Dr\.|Cllr\.|Lady\.|Lord\.)'


    cleaned_text_titles = re.sub(pattern_titles_comma, '', text_input, flags=re.IGNORECASE)
    cleaned_text = re.sub(pattern_titles, '', cleaned_text_titles, flags=re.IGNORECASE)

    return cleaned_text.strip()
def check_name_and_entity(party_name, entity_value, dba = None):
    """
    Identify individuals by using the value "INDIVIDUAL" in the `<legal-entity-text>` tag name:
    - Remove common titles: ["Mr", "Mrs", "Miss", "Ms", "Mx", "Sir", "Dame", "Dr", "Cllr", "Lady", "Lord"].
    - These titles will mostly appear in the `party_name` at the BEGINNING and END. Define two patterns to address each

    Otherwise, consider it a corporation if the entity is not NA/UNKNOWNED:
    - Check for special tags ("&amp;" only).

    If the entity is NA/UNKNOWNED:
    - Check if any organizational pattern exists in the name or in <dba-aka-ta-statement> tagname.
    - If yes, keep the name as it is.
    - Otherwise, consider it an individual and remove titles.

    Return party_name and entity_value after validation
    """


    # Regular expression pattern to match individual-related terms
    indi_pattern = r'\b(?:individual)\b'

    # Regular expression pattern to match organization-related terms
```

```
    # Regular expression pattern to match organization related terms
    org_pattern = r'\b(?:ltd|llp|llc|BV|B\.V\.|company|corporation|cor|bank|banking|inc|limited|sas|GMBH|S\.R\.L|SOCIÉTÉ

    # Clean party_name and entity_value
    party_name, entity_value = convert_character(party_name), convert_character(entity_value)

    # Indentify individual pattern in entity value and remove titles if it exists, ignoring case
    if re.search(indi_pattern, entity_value, re.IGNORECASE):
        party_name = remove_person_titles(party_name)
        return party_name, entity_value

    # If entity_value cannot be defined entity is individual or organization
      # search for organization pattern(characters in the name or sometime dba attribute) to classify organization,
      # if these signals can't be found then treat party_name as individual
    elif entity_value.upper() in ['NOT PROVIDED','UNKNOWN','NA','']:
        party_name = party_name.strip() if (re.search(org_pattern, party_name, re.IGNORECASE) or re.search(org_pattern,
        return party_name, entity_value

    # this one is corporation
    else:
        return party_name, entity_value
```

```
"""Print the code to check party_name and entity"""
# Extract information about individual, remove title DR
check1 = check_name_and_entity('ZYCHLINSKI ING. BRUNO, DR.','INDIVIDUAL', 'TA THE FOURTEEN HUNDRED COMPANY')
print(check1)

# Extract information about organization, in case of not knowing entity (check for organizational pattern in the name o
check2 = check_name_and_entity('ZYCHLINSKI ING. BRUNO, DR.','UNKNOWN', 'TA THE FOURTEEN HUNDRED COMPANY')
print(check2)
```

When combining all the above functions, we will successfully extract the detailed information of assignees and assignors, in which difference lies on the country's prioritized tagname list:

1. assignees: ['country-name', 'state', 'nationality','city']
2. assignors: ['nationality', 'country-name', 'state', 'city']

```
def assignees_info(text_info):
  """
  This function creates assignee information by extracting corresponding information from tagnames
     Prioritized tagnames: ['country-name', 'state', 'nationality','city']
    Extracted enformation also will be checked and transformed to desired format via above checking/converting functior
  Return will be assignees information in which each is packed in a dictionary and finally all will be included in a l
  """
  assignees_info = []
  split_data = tagname_info('assignee', text_info, False)

  for element in split_data:
    party_name = tagname_info('person-or-organization-name', element, True)
    country =  country_converted(['country-name', 'state', 'nationality','city'], element)
    # country = country_converted(['nationality', 'country-name', 'state', 'city'], element)
    legal_entity_text = tagname_info('legal-entity-text', element, True)
    dba = tagname_info('dba-aka-ta-statement', element, True)
    party_name, legal_entity_text = check_name_and_entity(party_name, legal_entity_text, dba)
    dic = {
        'party-name': party_name,
        'country': country,
        'legal-entity-text': legal_entity_text
    }
    assignees_info.append(dic)
  return assignees_info

def assignors_info(text_info):
  """
  This function creates assignors information by extracting corresponding information from tagnames
    Prioritized tag names: ['nationality', 'country-name', 'state', 'city']
    Extracted enformation also will be checked and transformed to desired format via above checking/converting functior
  Return will be assignors information in which each is packed in a dictionary and finally all will be included in a l
  """
  assignors_info = []
  split_data = tagname_info('assignor', text_info, False)

  for element in split_data:
    party_name = tagname_info('person-or-organization-name', element, True)
    date_acknowledged = date_converted('date-acknowledged', element)
```

```
    execution_date = date_converted('execution-date', element)
    country = country_converted(['nationality', 'country-name', 'state', 'city'], element)
    legal_entity_text = tagname_info('legal-entity-text', element, True)
    dba = tagname_info('dba-aka-ta-statement', element, True)
    party_name, legal_entity_text = check_name_and_entity(party_name, legal_entity_text, dba)
    dic = {
        'party-name': party_name,
        'date-acknowledged' : date_acknowledged,
        'execution-date': execution_date,
        'country': country,
        'legal-entity-text': legal_entity_text
    }
    assignors_info.append(dic)
  return assignors_info
```

This part shows the combination of all assignment's single attributes, assignors and assignees information under unique ID (**rf_id**). The final return will be assignment information in dictionary format.

```
def assignment_info(text_info):
  """
  This function returns all assignment information from text, using above functions
  """
  assignment_info = {}
  split_data = tagname_info('assignment', text_info, False)

  for element in split_data:
    rf_id = rf_id_converted('reel-no', 'frame-no', element)
    last_updated_date = date_converted('last-update-date', element)
    conveyance_text = tagname_info('conveyance-text', element, True)
    correspondent_party = tagname_info('person-or-organization-name', element, True)

    dic = {
        'last-update-date': last_updated_date,
        'conveyance-text' : conveyance_text,
        'correspondent-party': correspondent_party,
        'assignors-info': assignors_info(text_info),
        'assignees-info': assignees_info(text_info),
        'property-count': property_converted(text_info)
    }
    assignment_info[rf_id] = dic
  return assignment_info
```

```
# Final assignment information and print it in json format for easy reading
assignment_dic = assignment_info(raw_info)
print(json.dumps(assignment_dic, indent=4))
```

Lastly, we will create the json_file() function to read all the assignment information in a text file, append these into a dictionary and convert it into `JSON` format. This function will have 3 parameters:

1. **input_path**: input file's path
2. **output_path**: output file's path
3. **write_output**: write the result in output_path, the default value is False, meaning that no writing on the output_path

After that, we will demonstrate examples of both non-writing and writing output.

```
def json_file(input_path, output_path=None, write_output=False):
    """
    Read content from the input JSON file, extract transaction information, merge dictionaries, and write to output fi

    Parameters:
        input_path (str): Path to the input JSON file.
        output_path (str, optional): Path to the output JSON file. Defaults to None.
        write_output (bool, optional): Whether to write the output to a file. Defaults to False.

    Returns:
        str or None: JSON data if write_output is False, None otherwise.
    """
    with open(input_path, 'r') as input_file:
        content = input_file.read()

        # Final dictionary contains all assignment information
```

```
            merged_dict = {}
            # Extract list of assignment information under tag name assignment-entry
            transaction_info = tagname_info('assignment-entry', content, False)

            # Transform each assignment raw information and put it in final dictionary
            for element in transaction_info:
                element_in_dic = assignment_info(element)
                merged_dict.update(element_in_dic)

        if write_output and output_path:
            # Convert final dictionary into JSON data and write it to output file
            with open(output_path, 'w') as output_file:
                json.dump(merged_dict, output_file, indent=4)
            print(f"JSON file saved successfully at {output_path}")
        else:
            # Return the JSON data
            return json.dumps(merged_dict)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
input_path = '/content/drive/Shareddrives/FIT5196_S1_2024/A1/Sample/Input/sample_input_task1.txt'
assignment_json = json_file(input_path, None, False)
assignment_data = json.loads(assignment_json)
```

```
# Print all the values associated with the first key
first_key = next(iter(assignment_data))
print(json.dumps(assignment_data[first_key], indent=4))
```

5. Output verification and file extraction

5.1. Verification of the Generated JSON File

Here we do some validating between the sample output and our output on the sample input file.

```
with open('/content/drive/Shareddrives/FIT5196_S1_2024/A1/Sample/Output/sample_output_task1.json', 'r') as file1:
  sample_output_assignment_data = json.load(file1)
```

```
    def compare_attributes(data1, data2):
        # Get all keys present in both datasets
        all_keys = set(data1.keys()).union(data2.keys())
        # Compare attributes for each key
        for key in all_keys:
```

The main difference lies in the character '&' in our file being replaced by empty ('') in the sample output, which in this case the sample output is incorrect (there is no specification about replacing '&' character). Apart from that, there's not a lot of mismatch in the party's name and country output, which means our classification is mostly accurate.