DSE 203 - Knowledge Graph Report

NFL Players with Chronic Traumatic Encephalopathy (CTE)

Group #5 ● Ramona Henry, Alejandro Hohmann, Cammilus Perera ● DECEMBER, 2022

Contents

Section 1 Overview 3
1.1 Context 3
1.2 Objective 3
1.3 Scope of Work 3
Section 2 Dat Sources 4
Section 3 Data Sources 1 & 2 Inegration Stratergy 5
Section 4 Data Source 3 Named Entity Recognition(NER) & LDA Topic Modeling 11
Section 5 Data Sources 2 & 3 Inegration Stratergy 11
5.1 .... **Error! Bookmark not defined.**
Section 6 12
Section 7 12
Section 8 Revision History 12

Section 1: Overview

1.1 Context

Many former National Football League (NFL) players have been diagnosed with or have had chronic traumatic encephalopathy, or CTE. A definitive diagnosis so far can be made only post-mortem. However, an increasing number of former players are reporting symptoms of CTE.

According to 2017 study on brains of deceased gridiron football players, 99% of tested brains of NFL players, 88% of Canadian Football League (CFL) players, 64% of semi-professional players, 91% of college football players, and 21% of high school football players had various stages of CTE. However, this study had several limitations, including possible selection bias as families of players with symptoms of CTE are far more likely to donate brains to research than those without signs of the disease. Despite the limitations, the study still showed that CTE is far more common than once believed.

1.2 Objective

Organize data from multiple sources, about entities of interest associated with CTE in the NFL, and forge connections between them to represent a network of real-world entities—i.e. objects, events, situations, or concepts—and illustrates the relationship between them in a graph database and visualized as a graph structure.

1.3 Scope of Work

Integrate data from three different sources which combines a mixture of structured, semi-structured, and unstructured data to construct a knowledge graph per below provided project requirements for UCSD's DSE203 final project.

Project Requirements: *Your knowledge graph need not be humongous, but it should have a reasonable size (nodes + edges)*

- *100 is too low, 100k is high*
- *The number of edge labels should not be less than 7*
- *The number of node labels should not be less than 7*

Your methods should include some degree of information extraction from text and some degree of value matching

Try to make your use cases realistic. Your example queries on the knowledge graph

- *Should have some degree of complexity*
- *May include some analytical operations on graphs that are permitted by Neo4J (and Python if needed)*

Section 2: Data Sources

The project team combined a mixture of datasets from the following publicly available data sources.

Data source 1: NFL Statistics

***Note: The project team only utilized the Basic_Stats.csv dataset from data-source-1.*

Data Variety - Structured data.

Data Source URL: https://www.kaggle.com/datasets/kendallgillies/nflstatistics

Download URL: https://github.com/mona-jandro-camm/dse203/tree/main/Datasets

DDL Schema:

```
create table etl."Basic_Stats"
(
   "Age"                  integer,
   "Birth Place"          text,
   "Birthday"             text,
   "College"              text,
   "Current Status"       text,
   "Current Team"         text,
   "Experience"           text,
   "Height (inches)"      integer,
   "High School"          text,
   "High School Location" text,
   "Name"                 text,
   "Number"               integer,
   "Player Id"            text,
   "Position"             text,
   "Weight (lbs)"         integer,
   "Years Played"         text
);
```

Data source 2: NFL players with chronic traumatic encephalopathy

Data Variety - Semi structured and unstructured data.

Data Source URL:

https://en.wikipedia.org/wiki/List_of_NFL_players_with_chronic_traumatic_encephalopathy

Data source 3: US Newspaper Articles from UCSD's AWESOME Database

Data Variety- Structured, semi-structured, and unstructured data.

Table name: usnewspaper

Connection details:

host - awesome-hw.sdsc.edu

database name - postgres

DDL Schema:

```
create table usnewspaper
(
   news                    text,
   id                      serial primary key,
   collectiondate          date,
   title                   varchar(600),
   url                     varchar(600),
   publishdate             date,
   author                  text[],
   keywords                text[],
   src                     varchar(400),
   language                varchar(2),
   newsindex               tsvector
);
```

Section 3: Data Sources 1 & 2 Integration Strategy

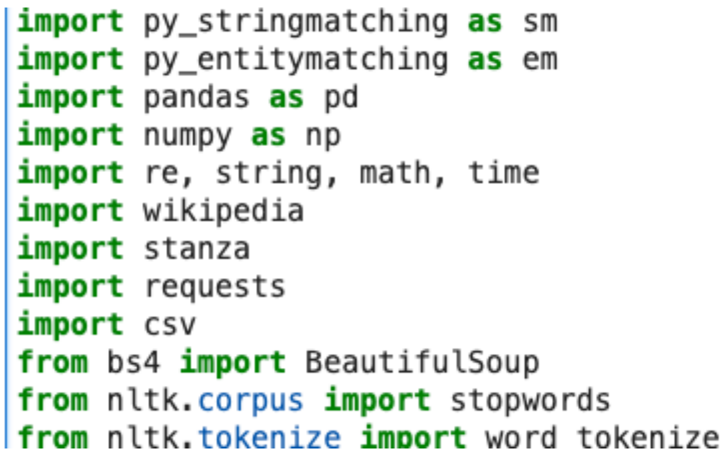
Method:

Extract NFL CTE affected player names from Wikipedia page and perform player name matching with NFL basic statistics dataset obtained from Kaggle. To extract the Wikipedia player names, the project team utilized Python's BeautifulSoup library, and for the name matching between both datasets, py_entitymatching's *attribute blocker EM process was utilized.*

Technology Stack:

The following technology stack was primarily used to integrate data sources 1 & 2.

- Python (Jupyter Notebook)
- SQL (Postgres SQL)
- Python Libraries

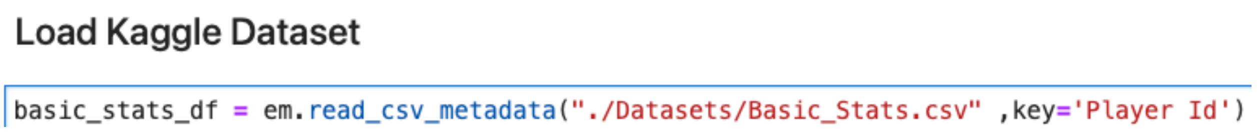


```python
import py_stringmatching as sm
import py_entitymatching as em
import pandas as pd
import numpy as np
import re, string, math, time
import wikipedia
import stanza
import requests
import csv
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

<u>Data Integration Steps:</u>

1. *Load structured CSV dataset from data source #1 - Kaggle into a DataFrame via py_entitymatching's read_csv_metadata method.*

   Sample Code Snippet:

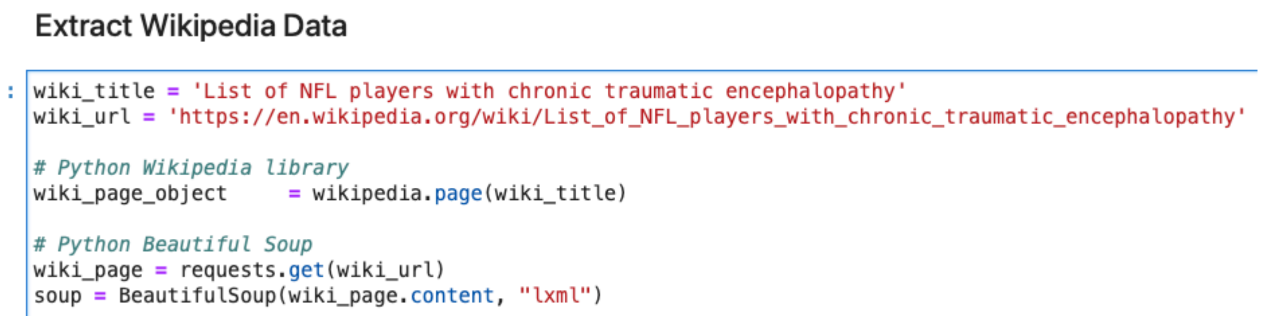


Load Kaggle Dataset

```
basic_stats_df = em.read_csv_metadata("./Datasets/Basic_Stats.csv" ,key='Player Id')
```

2. *Extract semi-structured and unstructured data from data source #2 - Wikipedia. The data extraction was done via Python's Wikipedia and BeautifulSoup libraries.*
   ***Note: BeautifulSoup provides additional drill-down functionalty to get at specific HTML tags in comparison to Python's Wikipedia library.*

   Sample Code Snippet:

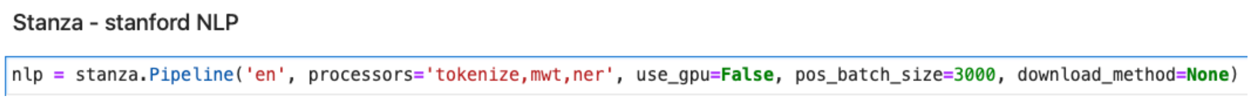


Extract Wikipedia Data

```
wiki_title = 'List of NFL players with chronic traumatic encephalopathy'
wiki_url = 'https://en.wikipedia.org/wiki/List_of_NFL_players_with_chronic_traumatic_encephalopathy'

# Python Wikipedia library
wiki_page_object     = wikipedia.page(wiki_title)

# Python Beautiful Soup
wiki_page = requests.get(wiki_url)
soup = BeautifulSoup(wiki_page.content, "lxml")
```

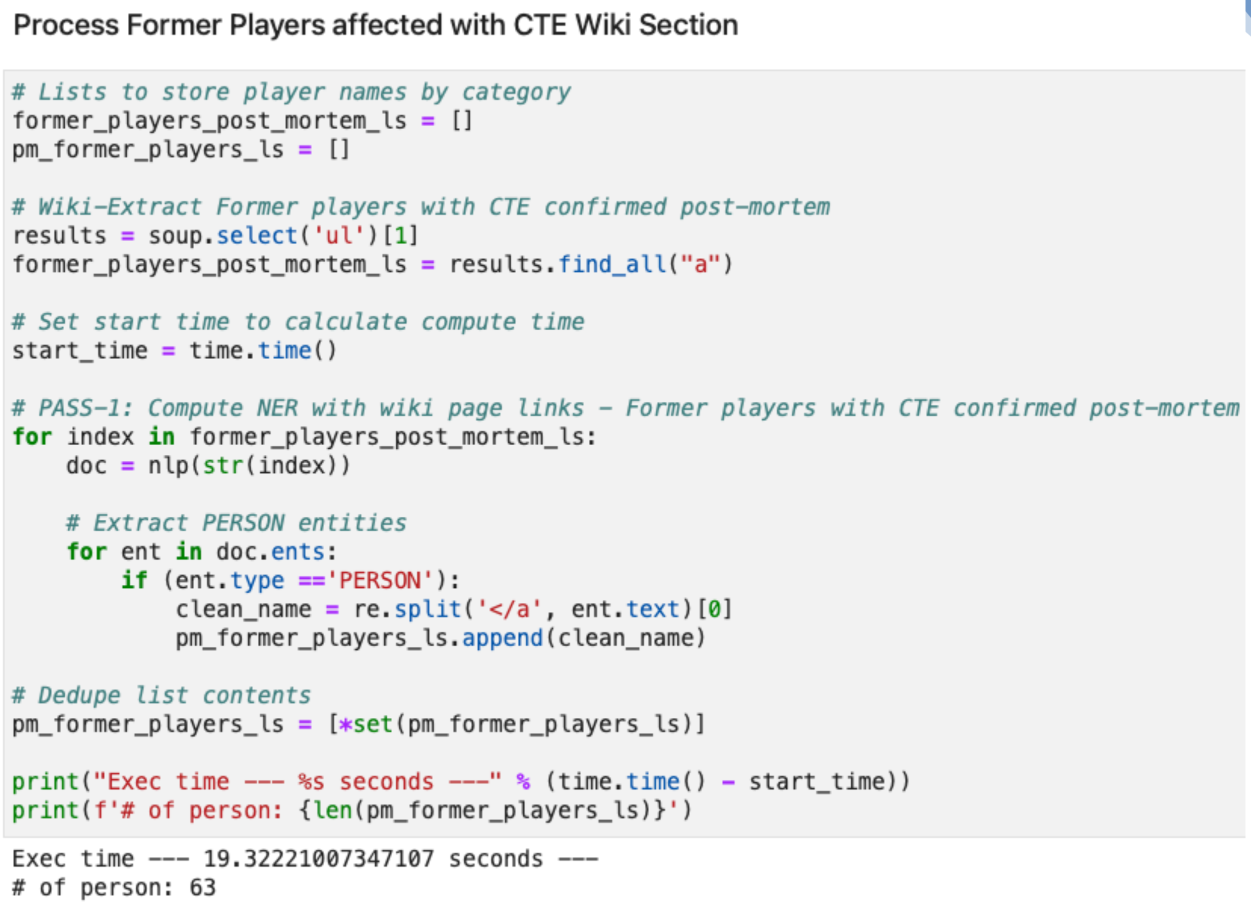
3. *Perform Named Entity Recognition (NER) and extract "PERSON" entities with the help of Python's Stanza library for data source #2 - Wikipedia. The "PERSON" entity data was placed into separate data/lists structures based on each header category they are listed in data source #2. Those five different "PERSON" header categories are listed below.*
   - *Players Affected Wiki Section*
   - *Former Players affected with CTE Wiki Section*
   - *Deceased players suspected of having had CTE Wiki Sction*
   - *Living former players diagnosed with CTE or ALS or reporting symptoms consistent with CTE or ALS Wiki Section*
   - *Former players listed as plaintiffs in lawsuits against the NFL for concussion-related injuries received after Wiki playing Section*

   Sample Code Snippet:



Stanza - stanford NLP

```
nlp = stanza.Pipeline('en', processors='tokenize,mwt,ner', use_gpu=False, pos_batch_size=3000, download_method=None)
```



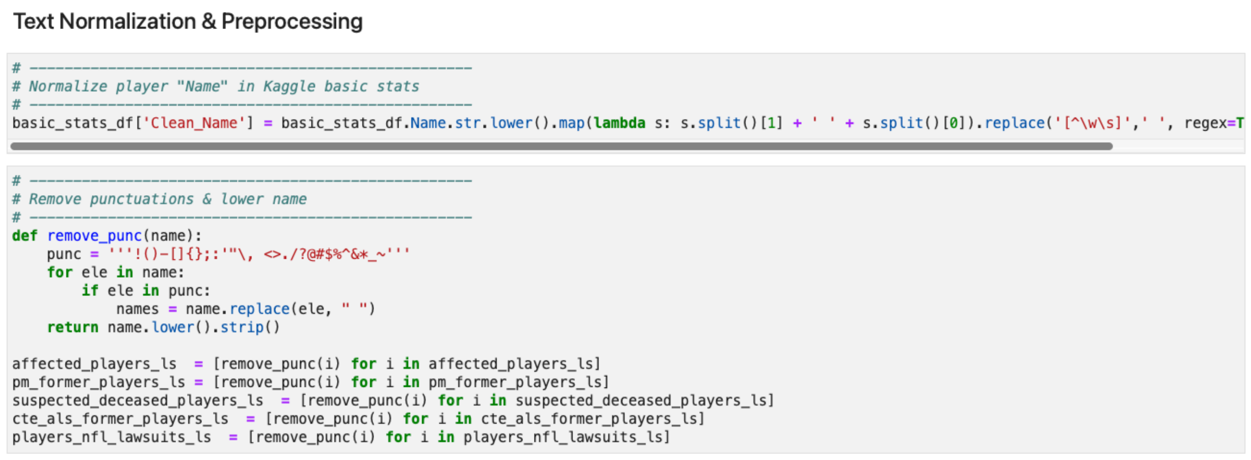
Process Former Players affected with CTE Wiki Section

```
# Lists to store player names by category
former_players_post_mortem_ls = []
pm_former_players_ls = []

# Wiki-Extract Former players with CTE confirmed post-mortem
results = soup.select('ul')[1]
former_players_post_mortem_ls = results.find_all("a")

# Set start time to calculate compute time
start_time = time.time()

# PASS-1: Compute NER with wiki page links - Former players with CTE confirmed post-mortem
for index in former_players_post_mortem_ls:
    doc = nlp(str(index))

    # Extract PERSON entities
    for ent in doc.ents:
        if (ent.type =='PERSON'):
            clean_name = re.split('</a', ent.text)[0]
            pm_former_players_ls.append(clean_name)

# Dedupe list contents
pm_former_players_ls = [*set(pm_former_players_ls)]

print("Exec time --- %s seconds ---" % (time.time() - start_time))
print(f'# of person: {len(pm_former_players_ls)}')
```

```
Exec time --- 19.32221007347107 seconds ---
# of person: 63
```

4. Text normalization and pre-processing to lower and remove punctuations from text.

Sample Code Snippet:



Text Normalization & Preprocessing

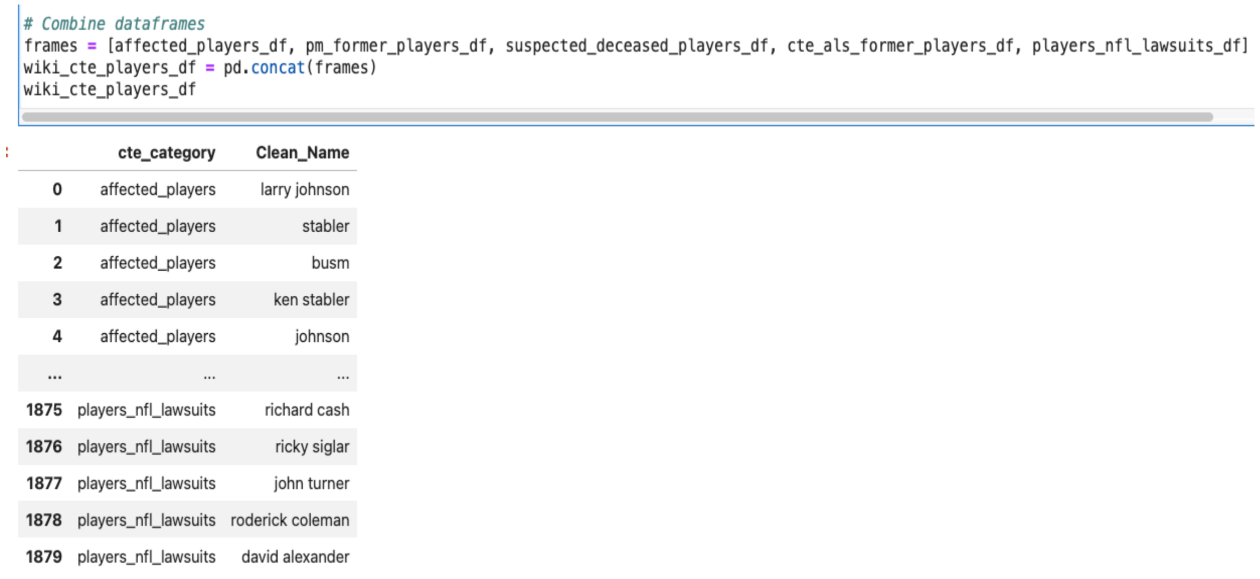
```
# ---------------------------------------------------
# Normalize player "Name" in Kaggle basic stats
# ---------------------------------------------------
basic_stats_df['Clean_Name'] = basic_stats_df.Name.str.lower().map(lambda s: s.split()[1] + ' ' + s.split()[0]).replace('[^\w\s]',' ', regex=T
```

```
# ---------------------------------------------------
# Remove punctuations & lower name
# ---------------------------------------------------
def remove_punc(name):
    punc = '''!()-[]{};:'"\, <>./?@#$%^&*_~'''
    for ele in name:
        if ele in punc:
            names = name.replace(ele, " ")
    return name.lower().strip()

affected_players_ls  = [remove_punc(i) for i in affected_players_ls]
pm_former_players_ls = [remove_punc(i) for i in pm_former_players_ls]
suspected_deceased_players_ls  = [remove_punc(i) for i in suspected_deceased_players_ls]
cte_als_former_players_ls  = [remove_punc(i) for i in cte_als_former_players_ls]
players_nfl_lawsuits_ls  = [remove_punc(i) for i in players_nfl_lawsuits_ls]
```

5. Combine the five different header categories data structures which contain NFL player names from data source #2 into one single DataFrame while creating following groups for each header category.

Wiki Header Section	**CTE Category Group**
Players Affected Wiki Section	affected_players
Former Players affected with CTE Wiki Section	pm_former_players
Deceased players suspected of having had CTE Wiki Sction	suspected_deceased_players
Living former players diagnosed with CTE or ALS or reporting symptoms consistent with CTE or ALS Wiki Section	cte_als_former_players
Former players listed as plaintiffs in lawsuits against the NFL for concussion-related injuries received after Wiki playing Section	players_nfl_lawsuits

Sample Code Snippet:

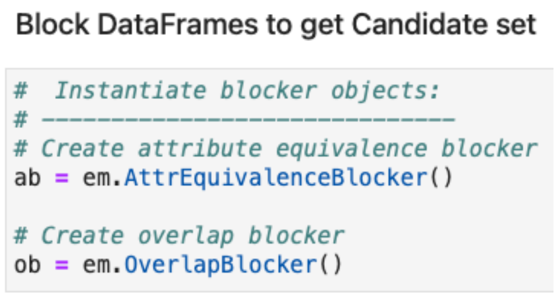


```
# Combine dataframes
frames = [affected_players_df, pm_former_players_df, suspected_deceased_players_df, cte_als_former_players_df, players_nfl_lawsuits_df]
wiki_cte_players_df = pd.concat(frames)
wiki_cte_players_df
```

```
           cte_category       Clean_Name
   0      affected_players    larry johnson
   1      affected_players          stabler
   2      affected_players             busm
   3      affected_players      ken stabler
   4      affected_players          johnson
 ...                   ...              ...
1875  players_nfl_lawsuits     richard cash
1876  players_nfl_lawsuits     ricky siglar
1877  players_nfl_lawsuits      john turner
1878  players_nfl_lawsuits roderick coleman
1879  players_nfl_lawsuits   david alexander
```

6. *Perform entity matching Attribute Blocking on data source #1 & #2 player names and create a new DataFrame.*

Sample Code Snippet:

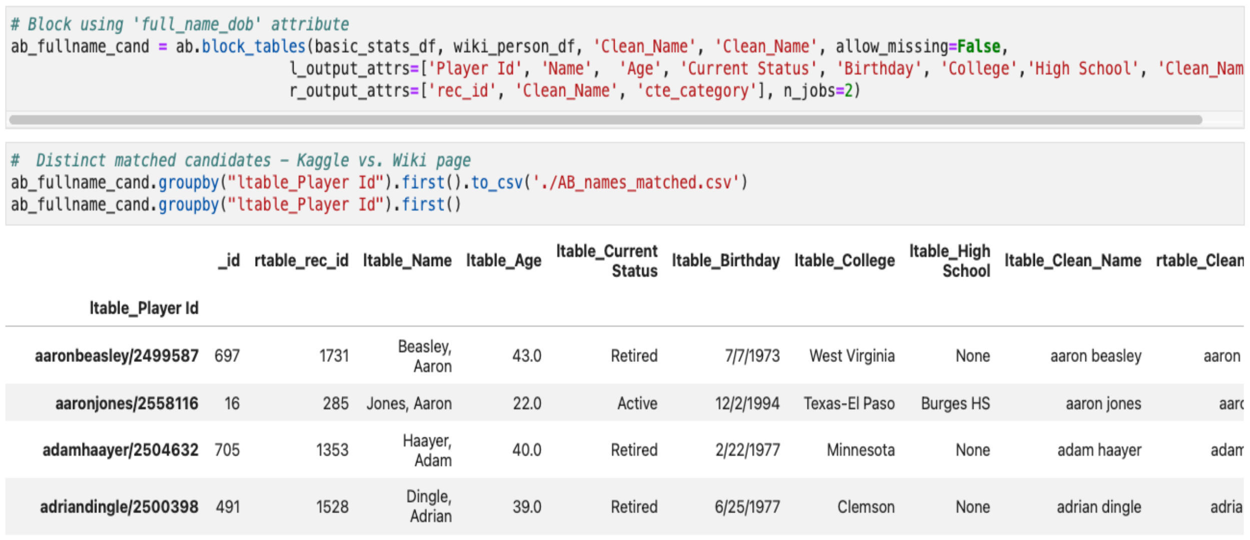


Block DataFrames to get Candidate set

```
#  Instantiate blocker objects:
# ------------------------------
# Create attribute equivalence blocker
ab = em.AttrEquivalenceBlocker()

# Create overlap blocker
ob = em.OverlapBlocker()
```

ii. Attribute Block by 'player_name'

***** BETTER RESULTS THAN OVERLAP BLOCK *****



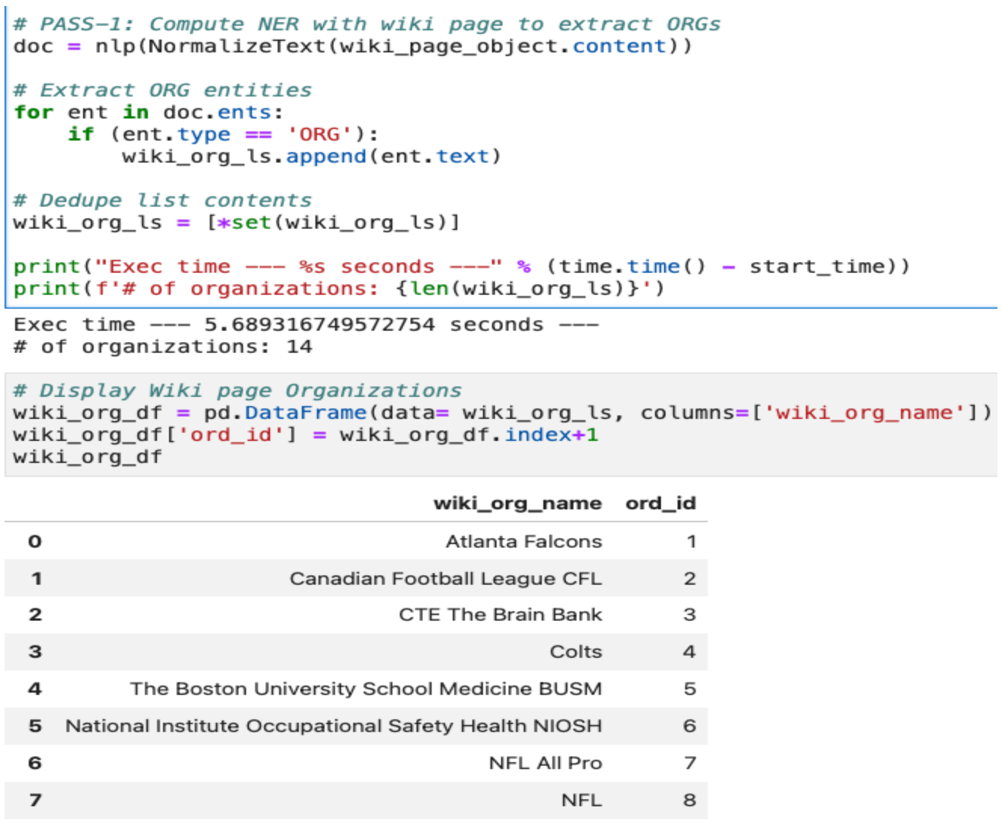
```
# Block using 'full_name_dob' attribute
ab_fullname_cand = ab.block_tables(basic_stats_df, wiki_person_df, 'Clean_Name', 'Clean_Name', allow_missing=False,
                                   l_output_attrs=['Player Id', 'Name',  'Age', 'Current Status', 'Birthday', 'College','High School', 'Clean_Nam
                                   r_output_attrs=['rec_id', 'Clean_Name', 'cte_category'], n_jobs=2)
```

```
#  Distinct matched candidates - Kaggle vs. Wiki page
ab_fullname_cand.groupby("ltable_Player Id").first().to_csv('./AB_names_matched.csv')
ab_fullname_cand.groupby("ltable_Player Id").first()
```

	_id	rtable_rec_id	ltable_Name	ltable_Age	ltable_Current Status	ltable_Birthday	ltable_College	ltable_High School	ltable_Clean_Name	rtable_Clean
ltable_Player Id										
aaronbeasley/2499587	697	1731	Beasley, Aaron	43.0	Retired	7/7/1973	West Virginia	None	aaron beasley	aaron
aaronjones/2558116	16	285	Jones, Aaron	22.0	Active	12/2/1994	Texas-El Paso	Burges HS	aaron jones	aarc
adamhaayer/2504632	705	1353	Haayer, Adam	40.0	Retired	2/22/1977	Minnesota	None	adam haayer	adam
adriandingle/2500398	491	1528	Dingle, Adrian	39.0	Retired	6/25/1977	Clemson	None	adrian dingle	adria

7. *Perform Named Entity Recognition (NER) and extract "ORG" entities with the help of Python's Stanza library for data source #2 - Wikipedia. The "ORG" entity data was placed into separate DataFrame in order to map NFL player names to an organization entity.*

   Sample Code Snippet:



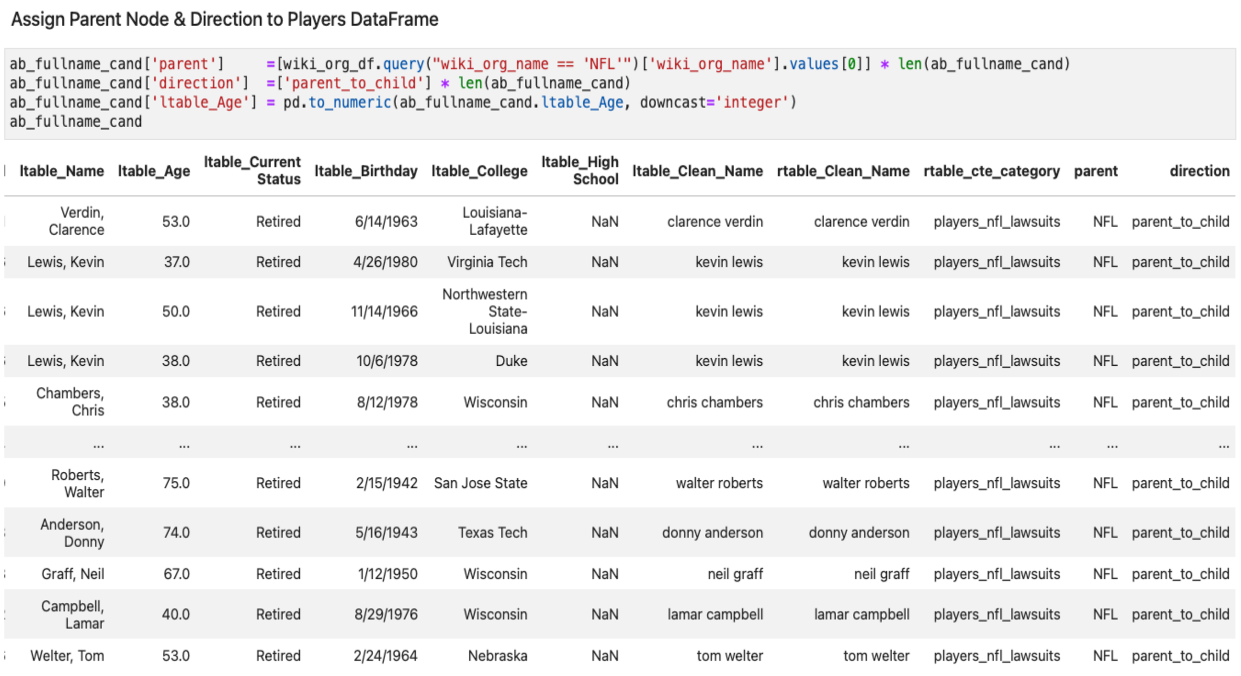
```
# PASS-1: Compute NER with wiki page to extract ORGs
doc = nlp(NormalizeText(wiki_page_object.content))

# Extract ORG entities
for ent in doc.ents:
    if (ent.type == 'ORG'):
        wiki_org_ls.append(ent.text)

# Dedupe list contents
wiki_org_ls = [*set(wiki_org_ls)]

print("Exec time --- %s seconds ---" % (time.time() - start_time))
print(f'# of organizations: {len(wiki_org_ls)}')
```

```
Exec time --- 5.689316749572754 seconds ---
# of organizations: 14
```

```
# Display Wiki page Organizations
wiki_org_df = pd.DataFrame(data= wiki_org_ls, columns=['wiki_org_name'])
wiki_org_df['ord_id'] = wiki_org_df.index+1
wiki_org_df
```

	wiki_org_name	ord_id
0	Atlanta Falcons	1
1	Canadian Football League CFL	2
2	CTE The Brain Bank	3
3	Colts	4
4	The Boston University School Medicine BUSM	5
5	National Institute Occupational Safety Health NIOSH	6
6	NFL All Pro	7
7	NFL	8

8. The blocked/matched NFL player names from data sources #1 & #2 are mapped to their parent "ORG" entity, i.e. NFL, and the graph model direction, i.e. partent-to-child, attribute is added to DataFrame created in step #6.
   Sample Code Snippet:

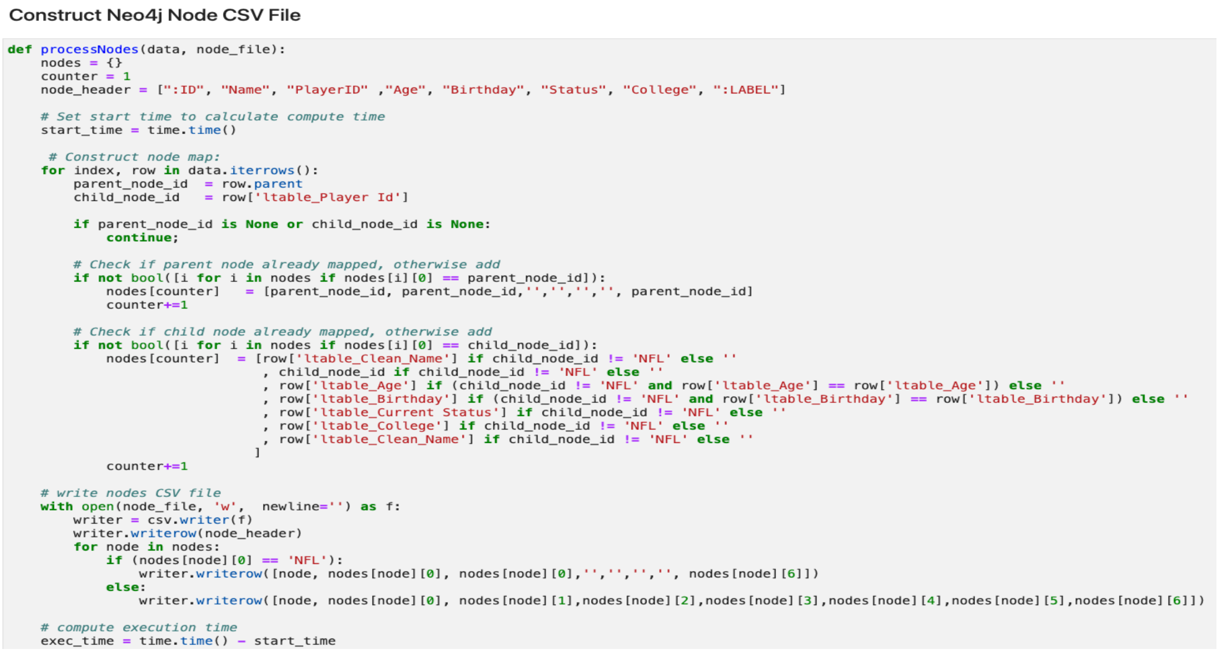


Assign Parent Node & Direction to Players DataFrame

```
ab_fullname_cand['parent']     =[wiki_org_df.query("wiki_org_name == 'NFL'")['wiki_org_name'].values[0]] * len(ab_fullname_cand)
ab_fullname_cand['direction']  =['parent_to_child'] * len(ab_fullname_cand)
ab_fullname_cand['ltable_Age'] = pd.to_numeric(ab_fullname_cand.ltable_Age, downcast='integer')
ab_fullname_cand
```

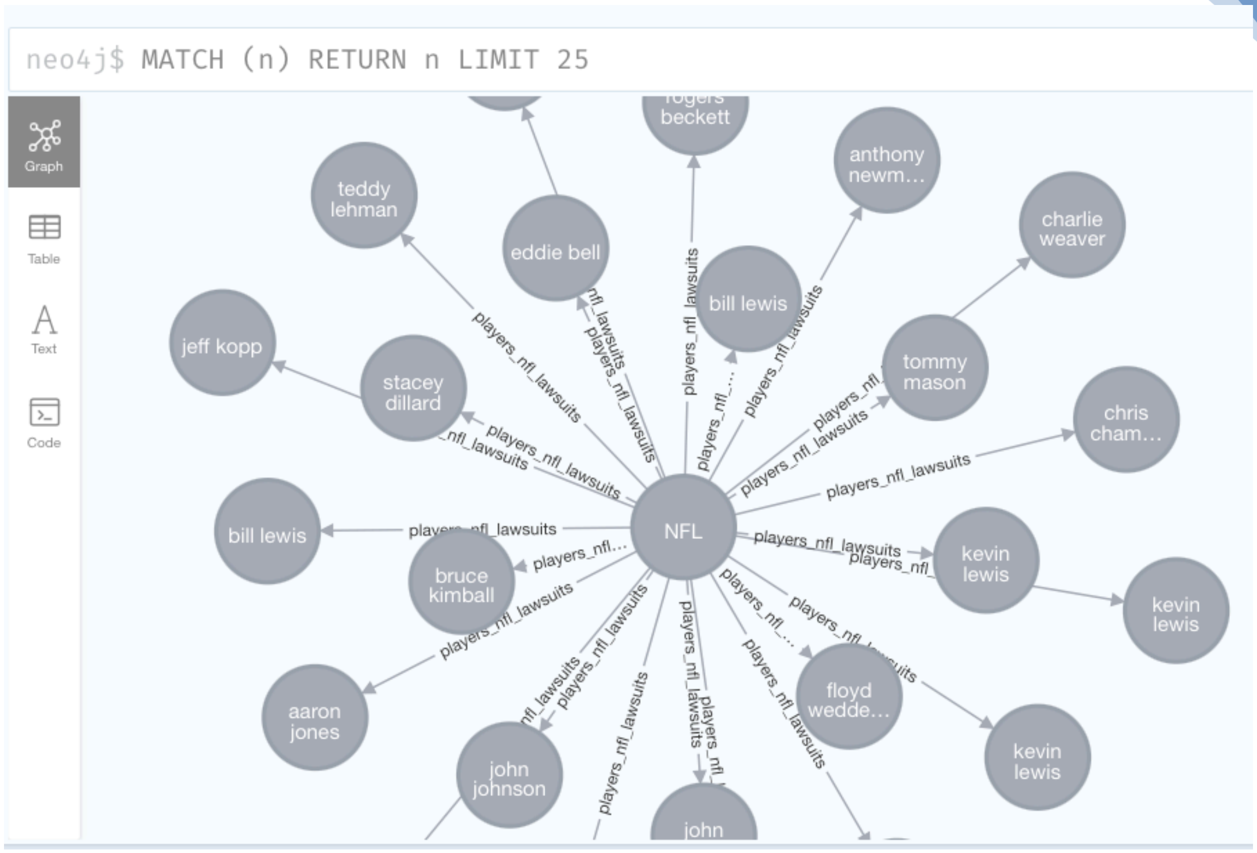
ltable_Name	ltable_Age	ltable_Current Status	ltable_Birthday	ltable_College	ltable_High School	ltable_Clean_Name	rtable_Clean_Name	rtable_cte_category	parent	direction
Verdin, Clarence	53.0	Retired	6/14/1963	Louisiana-Lafayette	NaN	clarence verdin	clarence verdin	players_nfl_lawsuits	NFL	parent_to_child
Lewis, Kevin	37.0	Retired	4/26/1980	Virginia Tech	NaN	kevin lewis	kevin lewis	players_nfl_lawsuits	NFL	parent_to_child
Lewis, Kevin	50.0	Retired	11/14/1966	Northwestern State-Louisiana	NaN	kevin lewis	kevin lewis	players_nfl_lawsuits	NFL	parent_to_child
Lewis, Kevin	38.0	Retired	10/6/1978	Duke	NaN	kevin lewis	kevin lewis	players_nfl_lawsuits	NFL	parent_to_child
Chambers, Chris	38.0	Retired	8/12/1978	Wisconsin	NaN	chris chambers	chris chambers	players_nfl_lawsuits	NFL	parent_to_child
...	...	...	...	...	...	...	...	...	...	...
Roberts, Walter	75.0	Retired	2/15/1942	San Jose State	NaN	walter roberts	walter roberts	players_nfl_lawsuits	NFL	parent_to_child
Anderson, Donny	74.0	Retired	5/16/1943	Texas Tech	NaN	donny anderson	donny anderson	players_nfl_lawsuits	NFL	parent_to_child
Graff, Neil	67.0	Retired	1/12/1950	Wisconsin	NaN	neil graff	neil graff	players_nfl_lawsuits	NFL	parent_to_child
Campbell, Lamar	40.0	Retired	8/29/1976	Wisconsin	NaN	lamar campbell	lamar campbell	players_nfl_lawsuits	NFL	parent_to_child
Welter, Tom	53.0	Retired	2/24/1964	Nebraska	NaN	tom welter	tom welter	players_nfl_lawsuits	NFL	parent_to_child

9. Construct and unit-test Neo4j knowledge graph to for data source #1 & #2.
   Basic Rules logic: The nodes are simply players and their parent "ORG" entity NFL.
   The edges are the five different CTE header categories from data source #2.
   Sample Code Snippet:



Construct Neo4j Node CSV File

```
def processNodes(data, node_file):
    nodes = {}
    counter = 1
    node_header = [":ID", "Name", "PlayerID" ,"Age", "Birthday", "Status", "College", ":LABEL"]

    # Set start time to calculate compute time
    start_time = time.time()

     # Construct node map:
    for index, row in data.iterrows():
        parent_node_id  = row.parent
        child_node_id   = row['ltable_Player Id']

        if parent_node_id is None or child_node_id is None:
            continue;

        # Check if parent node already mapped, otherwise add
        if not bool([i for i in nodes if nodes[i][0] == parent_node_id]):
            nodes[counter]   = [parent_node_id, parent_node_id,'','','','', parent_node_id]
            counter+=1

        # Check if child node already mapped, otherwise add
        if not bool([i for i in nodes if nodes[i][0] == child_node_id]):
            nodes[counter]  = [row['ltable_Clean_Name'] if child_node_id != 'NFL' else ''
                              , child_node_id if child_node_id != 'NFL' else ''
                              , row['ltable_Age'] if (child_node_id != 'NFL' and row['ltable_Age'] == row['ltable_Age']) else ''
                              , row['ltable_Birthday'] if (child_node_id != 'NFL' and row['ltable_Birthday'] == row['ltable_Birthday']) else ''
                              , row['ltable_Current Status'] if child_node_id != 'NFL' else ''
                              , row['ltable_College'] if child_node_id != 'NFL' else ''
                              , row['ltable_Clean_Name'] if child_node_id != 'NFL' else ''
                             ]
            counter+=1

    # write nodes CSV file
    with open(node_file, 'w',  newline='') as f:
        writer = csv.writer(f)
        writer.writerow(node_header)
        for node in nodes:
            if (nodes[node][0] == 'NFL'):
                writer.writerow([node, nodes[node][0], nodes[node][0],'','','','', nodes[node][6]])
            else:
                writer.writerow([node, nodes[node][0], nodes[node][1],nodes[node][2],nodes[node][3],nodes[node][4],nodes[node][5],nodes[node][6]])

    # compute execution time
    exec_time = time.time() - start_time
```



```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

Section 4: Data Source 3 Named Entity Recognition (NER) & LDA Topic Modeling

Describe the steps.

⇒

Section 5 Data Sources 2 & 3 Integration Strategy

This, and the following sections, define and explain each step in the integration process.

⇒

Section 6 Knowledge Graph Implementation Strategy

Describe the implementation strategy for the application.

Section 7 Analytical Queries

Describe the queries.

Section 8 Revision History

Identify changes

Version	Date	Name	Description