DOCUMENTAZIONE

TRACCIA 2 – CASO D'USO 1

Giuseppe Buonomano, Angelo Barletta
M63001506 – M63001507

Sommario

INTRODUZIONE

L'interesse nell'estrarre informazioni da testi non strutturati, come documenti clinici, articoli scientifici, notizie e post sui social media, è aumentato significativamente, richiedendo tecniche avanzate di Elaborazione del Linguaggio Naturale (NLP). Il Riconoscimento di Entità Nominate (NER) e l'Estrazione di Relazioni (RE) mirano a estrarre e collegare le entità identificate nel testo. Queste tecniche aiutano a tradurre informazioni complesse in un Grafo della Conoscenza (KG), dove le entità sono nodi e le relazioni tra loro sono archi. L'Intelligenza Artificiale Generativa (GenAI) può migliorare l'efficienza e l'accuratezza di questi processi grazie alla capacità di comprendere il contesto e il significato dei dati testuali.

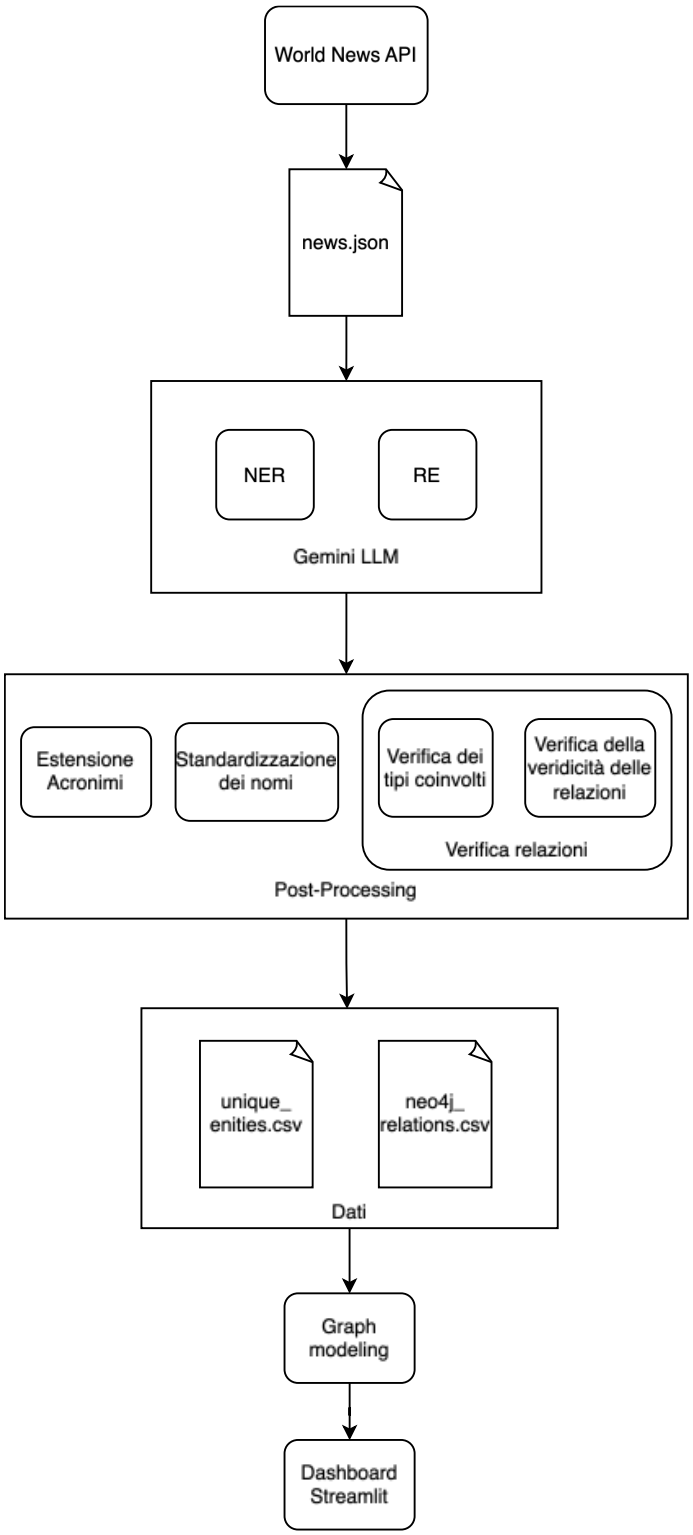
L'obiettivo del progetto è sviluppare un sistema per eseguire NER ed RE utilizzando le capacità di un Large Language Model (LLM). Il sistema deve identificare e collegare entità per costruire un KG e fornire un'analisi utile dei dati. L'architettura metodologica include i seguenti moduli:

- Dati Grezzi: Testo non strutturato (news), possono essere estratti da fonti esterne tramite API.
- Pre-elaborazione dei Dati: Pulizia, filtraggio e trasformazione dei dati per prepararli all'input nel LLM.
- Large Language Model (LLM): Un modello di intelligenza artificiale addestrato per generare testo naturale simile a quello scritto dagli esseri umani. In questo progetto, l'LLM estrae entità e le collega dai dati non strutturati seguendo questi passaggi:
    - Riconoscimento di Entità Nominate (NER): Identificazione e classificazione di segmenti di informazioni nel testo in categorie predefinite.
    - Estrazione di Relazioni (RE): Estrazione delle relazioni semantiche tra le entità riconosciute nel testo.
- Modellazione del Grafo: Le entità e le relazioni estratte serviranno come base per costruire una struttura di dati a grafo.
- Analisi dei Dati: Grazie al grafo costruito vengono eseguite una serie di analitiche sui dati.
- Report e Dashboard: Una dashboard sarà fornita come output per facilitare l'esplorazione dei dati da parte degli utenti.

Utilizzando i concetti estratti dall'LLM e integrati in un grafico della conoscenza, è possibile ottenere approfondimenti sulla semantica e le connessioni intrinseche nei dati. Il sistema permetterà agli utenti di ottenere

approfondimenti attraverso una dashboard intuitiva che supporta l'esplorazione e la visualizzazione dei dati raccolti, migliorando così la praticità e l'efficienza del sistema in scenari reali.

FLUSSO DI ESECUZIONE



RECUPERO NEWS

In questa fase, il sistema si concentra sul recupero di notizie politiche riguardanti le elezioni europee utilizzando le API di WorldNewsAPI.

Viene configurata l'API di WorldNewsAPI con una chiave API valida e vengono specificati i parametri di ricerca, come la parola chiave "Elezioni europee", la lingua italiana, e una fonte di notizie specifica (ad esempio, quelle da noi utilizzate sono: repubblica.it, ilmattino.it, ilmessaggero.it, ilsole24ore.com, iltempo.it, liberoquotidiano.it, lastampa.it, notiziegeopolitiche.net). Sono state utilizzate diverse fonti sia perché era nostro interesse recuperare il maggior numero di dati possibili, sia perché per ogni fonte non era possibile recuperare notizie più vecchie di un mese (ad esempio se uso l'API il 9 luglio, non posso recuperare una notizia dell'8 giugno).
Si imposta anche un intervallo di date e si stabilisce un limite al numero di notizie da recuperare.

Il sistema invia quindi una richiesta all'API per ottenere le notizie che corrispondono ai criteri specificati. Se la richiesta ha successo, le notizie vengono recuperate e salvate in un file JSON. Se esistono già notizie salvate da precedenti esecuzioni del programma, queste vengono lette e combinate con le nuove notizie recuperate.

Per la preparazione dei dati alla fase successiva, è stato eseguito un pre-processing consistente nella suddivisione delle news in chunk di dimensioni minori. Questo è necessario sia per adattarsi alle limitazioni del modello di linguaggio utilizzato successivamente nel progetto, che per evitare allucinazioni da parte del modello. Questa suddivisione avviene attraverso una funzione che segmenta il testo delle notizie in base alla lunghezza massima specificata. In particolare, il chunking viene fatto con sovrapposizione, l'idea è che in questo modo evitiamo di perdere informazioni necessarie all'identificazione di entità o relazioni all'interno delle frasi.

Infine, tutte le notizie processate e suddivise in chunk vengono salvate in un file JSON (news.json). Inoltre, viene aggiornato un offset per tenere traccia delle nuove notizie recuperate durante la prossima esecuzione del sistema. Questo processo assicura che il database delle notizie sia sempre aggiornato, estendibile e pronto per le successive fasi di analisi e modellazione.

NER & RE

Questa fase del progetto è dedicata all'interrogazione di un modello di linguaggio generativo per estrarre entità e relazioni dalle notizie raccolte, con l'obiettivo finale di costruire un grafo della conoscenza.

Viene inizialmente configurato il modello "gemini-1.5-flash", specificando una chiave API e le impostazioni di generazione. Il modello scelto viene configurato con parametri come temperatura (controlla la creatività o casualità della generazione del testo), top_p (gestisce la varietà e coerenza dell'output), top_k (limita la generazione del testo alle k parole con la probabilità più alta) e il numero massimo di token di output, per garantire una generazione di testo ottimale. Per avere un output da cui possiamo recuperare facilmente le informazioni che ci fornisce il modello, abbiamo scelto di imporre come tipo di risposta un file JSON.

A questo punto abbiamo indicato al modello il suo ruolo all'interno del campo system_instruction inserendo una serie di istruzioni per eseguire al meglio le operazioni di individuazione delle entità e relazioni dalle notizie.
In particolare, sono stati indicati tutti i possibili tipi delle entità e delle relazioni inserendo degli esempi e delle descrizioni per ognuno di essi.
Le entità sono delle seguenti tipologie:

- **Person**: Un individuo umano specifico, generalmente identificato per nome. Può includere nomi di persone famose, autori, artisti, scienziati, sportivi, ecc. Esempi: "Albert Einstein", "Leonardo da Vinci", "Marie Curie"

- **Politician**: Un individuo coinvolto in attività politiche, spesso eletto o nominato a una carica governativa. Può includere presidenti, primi ministri, senatori, sindaci e altri funzionari pubblici. Esempi: "Angela Merkel", "Barack Obama", "Giuseppe Conte"

- **Organization**: Un gruppo strutturato di persone che lavorano insieme per un obiettivo comune. Può includere aziende, enti governativi, istituzioni educative, ONG, gruppi sportivi, ecc. Esempi: "Google", "Nazioni Unite", "Università di Oxford"

- **Party (Partito)**: Un'organizzazione politica che rappresenta un gruppo di persone con ideologie e obiettivi comuni, solitamente con l'intento di ottenere e mantenere il potere politico attraverso elezioni. Esempi: "Partito Democratico", "Movimento 5 Stelle", "Partito Repubblicano"

- **Agreement (Accordo)**: Un'intesa formalmente riconosciuta tra due o più parti, che stabilisce diritti e doveri reciproci. Esempi: "Accordo di Parigi", "Trattato di Maastricht", "Contratto di lavoro collettivo"

- **Location**: Un luogo geografico specifico, che può essere una città, uno stato, un continente, un punto di interesse, ecc. Può includere anche indirizzi specifici, aree naturali e strutture. Esempi: "Roma", "Monte Everest", "Stati Uniti", "Piazza San Marco"

- **Event (Evento)**: Un'occasione particolare, spesso significativa, che può essere pianificata o spontanea. Include conferenze, guerre, catastrofi naturali, festival, eventi sportivi e celebrazioni. Esempi: "Giochi Olimpici", "Guerra Civile Americana", "Conferenza sul Clima di Parigi", "Festival di Cannes"

Le relazioni devono essere di questi tipi:

- **leader_of**: rapporto tra i leader politici e le entità o i gruppi che guidano. Fondamentale per comprendere la leadership e le affiliazioni politiche. Esempio: leader_of, Giorgia Meloni, Fratelli d'Italia

- **is_from**: Indica l'origine nazionale degli individui coinvolti in politica, importante per geolocalizzare i personaggi politici. Esempio: is_from, Giorgia Meloni, Italia

- **part_of**: mostra l'appartenenza di entità (come paesi, organizzazioni, dipartimenti) a gruppi più ampi come l'Unione Europea o il Parlamento Europeo. Esempio: part_of, Italia, G7

- **located_in**: specifica la posizione geografica di entità politiche o geografiche, utile per mappare la posizione di istituzioni ed eventi. Esempio: located_in, Colosseo, Italia

- **member_of**: indica l'appartenenza di individui (come politici o persone) a gruppi o organizzazioni politiche, fornendo approfondimenti sulle alleanze politiche. Esempio: member_of, Giorgia Meloni, Fratelli d'Italia

- **supports**: rappresenta il sostegno politico dato da una figura o organizzazione a un'altra, rivelando alleanze e coalizioni. Esempio: supports, Matteo Salvini, Giorgia Meloni

- **president_of**: identifica i soggetti che ricoprono la carica di presidente di enti politici o istituzionali. Esempio: president_of, Sergio Mattarella, Repubblica Italiana

- **opposition**: Indica opposizione politica tra figure o gruppi, utile per comprendere le dinamiche dei conflitti politici. Esempio: opposition, Giuseppe Conte, Giorgia Meloni

- **colleague**: Indica i rapporti professionali tra individui in politica, evidenziando collaborazioni e collegamenti. Esempio: colleague, Giuseppe Conte, Beppe Grillo

Ogni notizia di news.json viene passata al modello generativo in una sessione di chat. Per ognuna di esse vengono generate entità e relazioni che vengono salvate in un file JSON e poi estratte per essere salvate nei file CSV entities.csv e relations.csv, che contengono entità e relazioni di tutte le notizie.

Per poter riuscire ad utilizzare in maniera gratuita le API di Gemini, è necessario utilizzare una VPN, nel nostro caso ci siamo localizzati negli Stati Uniti.

POST PROCESSING

A questo punto, estratte le entità e relazioni di tutte le notizie, inizia una fase di post-processing relativa alla pulizia e al filtraggio dei dati ricavati.
Questa fase sarà composta da una serie di sottofasi.

ESTENSIONE ACRONIMI

La prima operazione che viene eseguita è quella di eliminazione di tutti i duplicati presenti nei file contenenti entità e relazioni, facendo attenzione al fatto che l'operazione non sia case sensitive.
Si verifica anche che tutti i tipi di entità e relazioni sono appartenenti a quelli che erano stati imposti al modello, se viene trovata un'entità/relazione non appartenente ai tipi precedentemente elencati, viene eliminata.

A questo punto viene nuovamente utilizzato il modello LLM Gemini che servirà ad analizzare tutte le entità precedentemente individuate in modo tale da creare una mappa di nomi (dizionario) in cui ad ogni acronimo (chiave) viene assegnato il suo nome esteso (valore). Ad esempio, l'entità "*M5S"* viene estesa in "*Movimento 5 Stelle",* in questo modo riusciamo a realizzare una sorta di standardizzazione degli acronimi, che viene poi estesa anche alle altre entità nella fase successiva.

STANDARDIZZAZIONE DEI NOMI

Dal momento che spesso nelle news ci si riferisce alla stessa entità in modi differenti, ad esempio "Alleanza Verdi e Sinistra" è la stessa entità di "Alleanza Verdi-Sinistra", è stato necessario verificare la similarità dei nomi per assegnarli alla stessa entità, questo è stato possibile grazie alla libreria python fuzzywuzzy, che utilizza tecniche di fuzzy matching per il confronto tra stringhe.
Il match tra le stringhe viene fatto sulla base di un valore di treshold, che noi abbiamo impostato ad 85. Quindi tutte le stringhe che hanno un punteggio di similitudine maggiore di 85 vengono trasformate nella stessa stringa finale.

Abbiamo inoltre notato che, per i nomi delle persone, esistevano alcune entità che presentavano sia nome che cognome, mentre altre riportavano solo uno dei due. Per risolvere questa inconsistenza, abbiamo effettuato un'ulteriore operazione di matching specifica per le entità di tipo "Person". In questo caso, abbiamo verificato se il campo "Name" di un'entità fosse incluso all'interno del campo "Name" di un'altra entità, mantenendo come risultato finale il nome completo (quello più lungo).

Ad esempio, il nome “Meloni” è diventato “Giorgia Meloni”, così come “Le Pen” è diventato “Marine Le Pen”.
L'obiettivo è creare un dizionario di sostituzione che mappa le varianti dei nomi a una forma standardizzata, migliorando così la coerenza e la qualità del dataset.

Il dizionario è stato poi applicato sia alle entità che alle relazioni.
Dopodiché sono stati eliminati tutti i duplicati per creare file senza ripetizioni.
L’output di questa fase sono i file *unique_entites.csv* e *unique_relations.csv*

VERIFICA RELAZIONI

I successivi passi sono stati di verifica della correttezza delle relazioni, sia per verificare che sia consentita quella specifica relazione per i tipi di entità coinvolti, che per verificare la veridicità della relazione stessa.

VERIFICA DEI TIPI COINVOLTI

Questa fase è utile a verificare che tutte le relazioni trovate in precedenza dal modello sono effettivamente possibili. A questo proposito abbiamo creato un file nominato *'relazioni_consentite.csv'* al cui interno sono stati elencati per ogni relazione quali sono i possibili tipi di entità coinvolti.
Alcune delle relazioni consentite sono:
leader_of,Person,Party
part_of,Party,Organization
located_in,Event,Location
member_of,Politician,Party
supports,Politician,Organization
president_of,Politician,Organization
opposition,Politician,Politician
colleague,Politician,Politician

Ad esempio, in una relazione come la seguente:
“leader_of, Italia, Unione Europea”
Il modello ha trovato una relazione di tipo “leader_of” tra due entità di tipo “Organization”, ciò non è consentito perché senza significato, di conseguenza bisogna eliminare questa relazione.
L’output di questa fase è il file *final_unique_relations.csv*

VERIFICA DELLA VERIDICITÀ

In quest'ultima fase di post-processing viene nuovamente interrogato il modello in modo tale da verificare che le relazioni trovate siano effettivamente corrette. Potrebbe infatti capitare che il modello inizialmente abbia trovato la seguente relazione: leader_of,Giuseppe Conte,Partito Democratico che seppur è consentita, è falsa, e va quindi eliminata.
Per istruire il modello ad eseguire questo compito, abbiamo inserito nel system_instruction nuovamente una descrizione di tutte le entità e delle relazioni coinvolte. Oltre a questo, è stato chiesto al modello di verificare la veridicità della relazione fornendo un output in formato JSON composto da una risposta SI/NO, e una motivazione per quest'ultima. L'aggiunta della motivazione nella risposta è utile a migliorarne la qualità.
Per ciascuna relazione l'output è composto dai campi: Answer, Relation, Source, Target e Motivazione.

Esempio1: "member_of, Berlusconi, Forza Italia" avrà il seguente output:

- "Answer": "SI",
- "Relation": "member_of",
- "Source": "Berlusconi",
- "Target": "Forza Italia",
- "Motivazione": "Silvio Berlusconi è stato leader di Forza Italia e quindi anche membro"

Esempio2: "member_of, Giorgia Meloni, Partito Democratico"

- "Answer": "NO",
- "Relation": "member_of",
- "Source": "Giorgia Meloni",
- "Target": "Partito Democratico",
- "Motivazione": "Giorgia Meloni è membro del partito Fratelli d'Italia e quindi non è membro del Partito Democratico"

Al modello sono state passate 25 relazioni alla volta a causa delle limitazioni sui token consentiti. Le risposte forniteci dal modello sono state salvate all'interno di un file JSON "*check_relations.json*".
Dopo aver processato tutte le relazioni abbiamo estratto dal JSON tutte le relazioni per le quali la risposta era "NO" ed eliminato le corrispondenti relazioni dal file precedentemente creato e contenente tutte le relazioni (*final_unique_relations.csv*) producendo in questo modo il file delle relazioni finali chiamato "*neo4j_relations.csv*" che verrà caricato sul database Neo4j insieme al file delle entità "*unique_entities.csv*".

CREAZIONE GRAPH DB

A partire dai file CSV prodotti in fase di pre-processing, adesso creiamo il database a grafo in neo4j.
Questo è possibile inserendo all'interno della cartella "*import*" i due file "*unique_entities.csv*" e "*neo4j_relations.csv*".
In particolare, è stato usato il plugin APOC per assegnare dinamicamente sia i nomi che le label a ciascuna entità e relazione.
Mostriamo di seguito il codice Cypher utilizzato per il caricamento di entità.

```
//Caricamento Entities
LOAD CSV WITH HEADERS FROM "file:///unique_entities.csv" as row
CALL apoc.merge.node([row.Type], {name: row.Name}) YIELD node
RETURN node
```

A questo punto è necessario accorpare le entità con lo stesso nome, perché nel dataset potrebbe essere presente la stessa entità più volte, ma con tipi differenti. Ad esempio, generalmente una persona può anche essere classificato come politico (es. Giorgia Meloni) oppure un'organizzazione può essere anche classificata come location (es. Italia). Quest'operazione viene fatta con il seguente codice Cypher:

```
// Trova tutti i nodi che hanno lo stesso nome
MATCH (n)
WITH n.name AS name, collect(n) AS nodes
WHERE size(nodes) > 1
// Unisci tutti i nodi con lo stesso nome
CALL apoc.refactor.mergeNodes(nodes, {properties: "discard", mergeRels: true})
YIELD node
RETURN count(*)
```

A questo punto è possibile creare le relazioni tra i nodi a partire dal file "*neo4j_relations.csv*". Ciò viene fatto con il seguente codice.

```
//Caricamento relazioni
LOAD CSV WITH HEADERS FROM "file:///neo4j_relations.csv" AS row
MATCH (startNode {name: row.Source})
MATCH (endNode {name: row.Target})
CALL apoc.create.relationship(startNode, row.Relation, {}, endNode) YIELD rel
RETURN rel
```

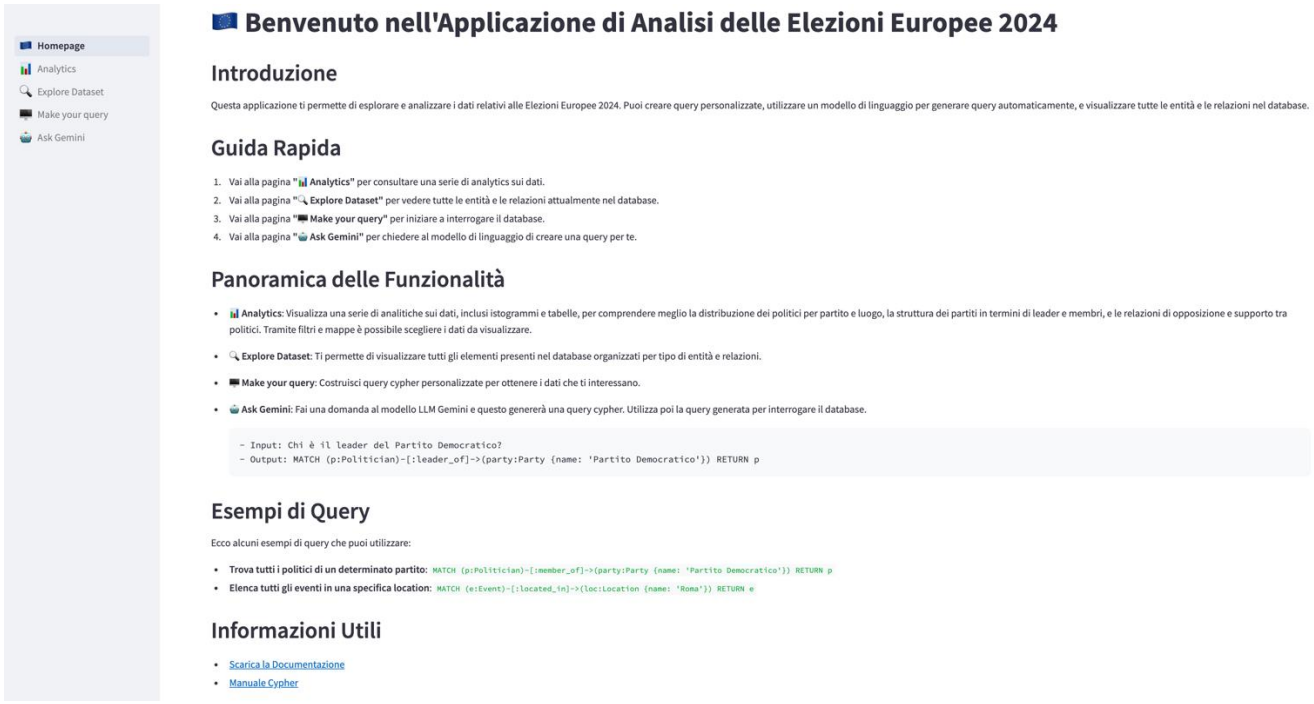
Per poter importare in maniera più semplice il dataset, senza il bisogno di effettuare nuovamente queste operazioni e senza la necessità di avere a disposizione i file CSV, abbiamo creato un dump file, da cui è possibile ricreare il nostro database in neo4j.
Il procedimento da seguire per caricare il database dal dump file è il seguente:

- Creare un nuovo progetto in neo4j
- Caricare il file .dump tramite “Add”->”File”
- Dalla sezione file del file .dump cliccare sul menù ⋮ e selezionare “Create New DBMS From Dump”

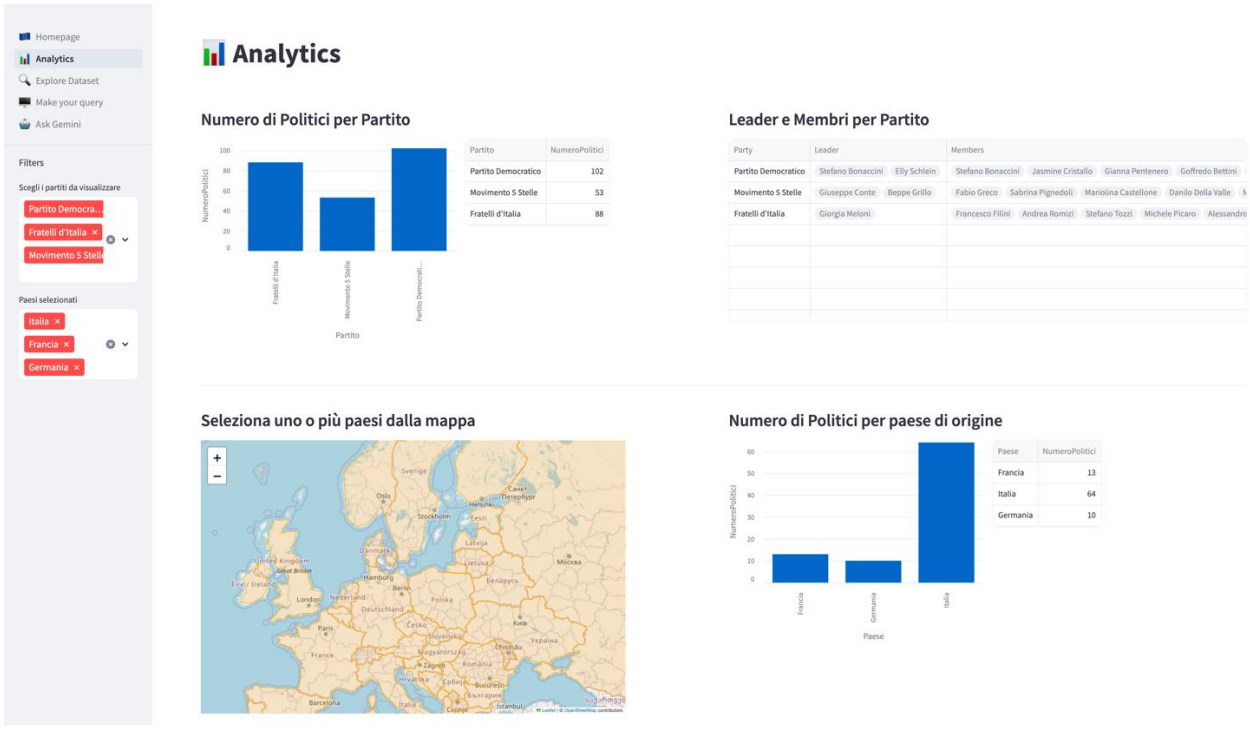
A questo punto il database è importato.

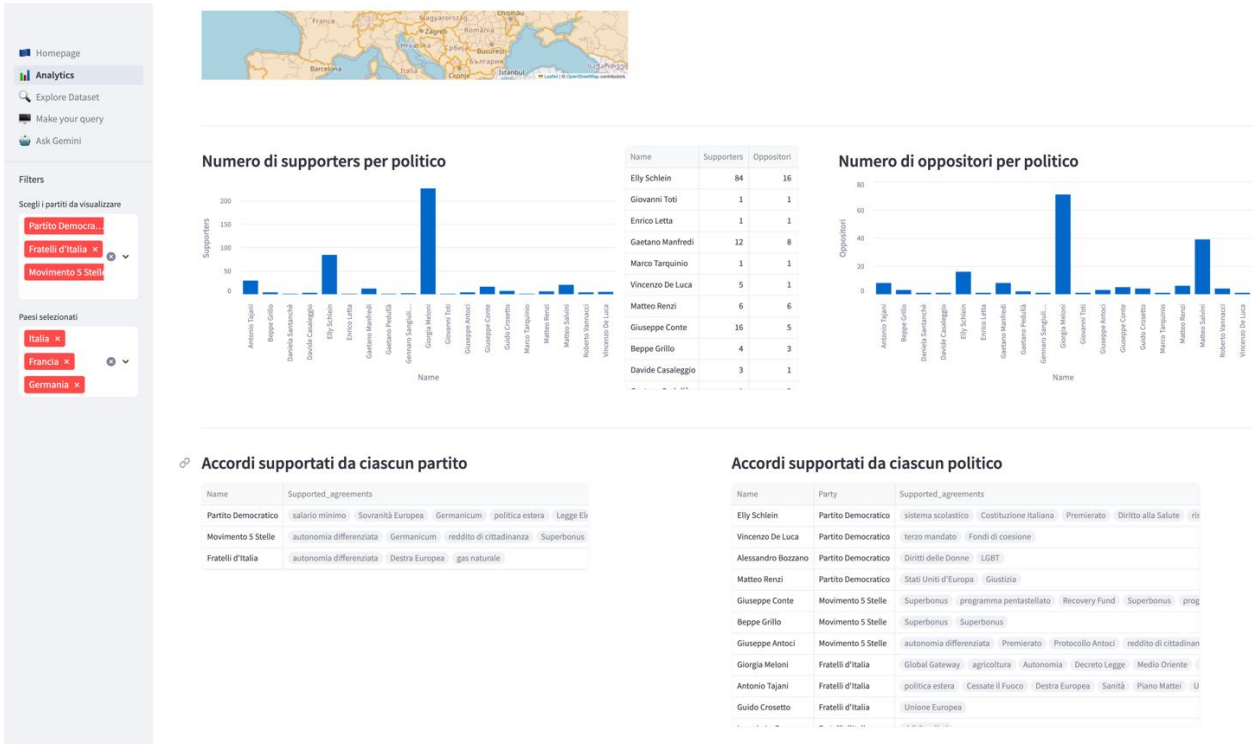
DASHBOARD STREAMLIT

Abbiamo creato una dashboard streamlit composta da più pagine, la prima è una homepage che spiega quali funzionalità includono le altre pagine

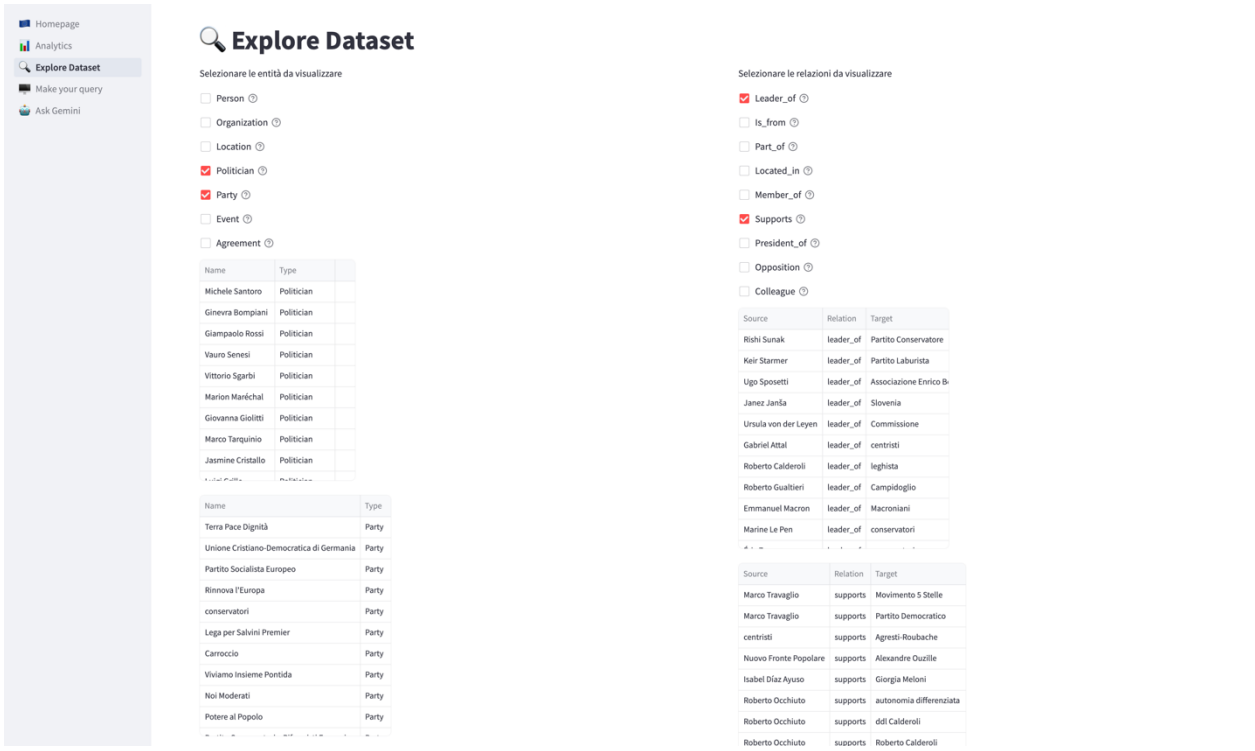


Nella pagina Analytics, vengono mostrate delle analisi eseguite sul graphDB. In particolare, è possibile personalizzare tali analisi agendo sui filtri presenti nella sidebar. Inoltre, è anche possibile utilizzare la mappa per selezionare i paesi da includere nel filtro.

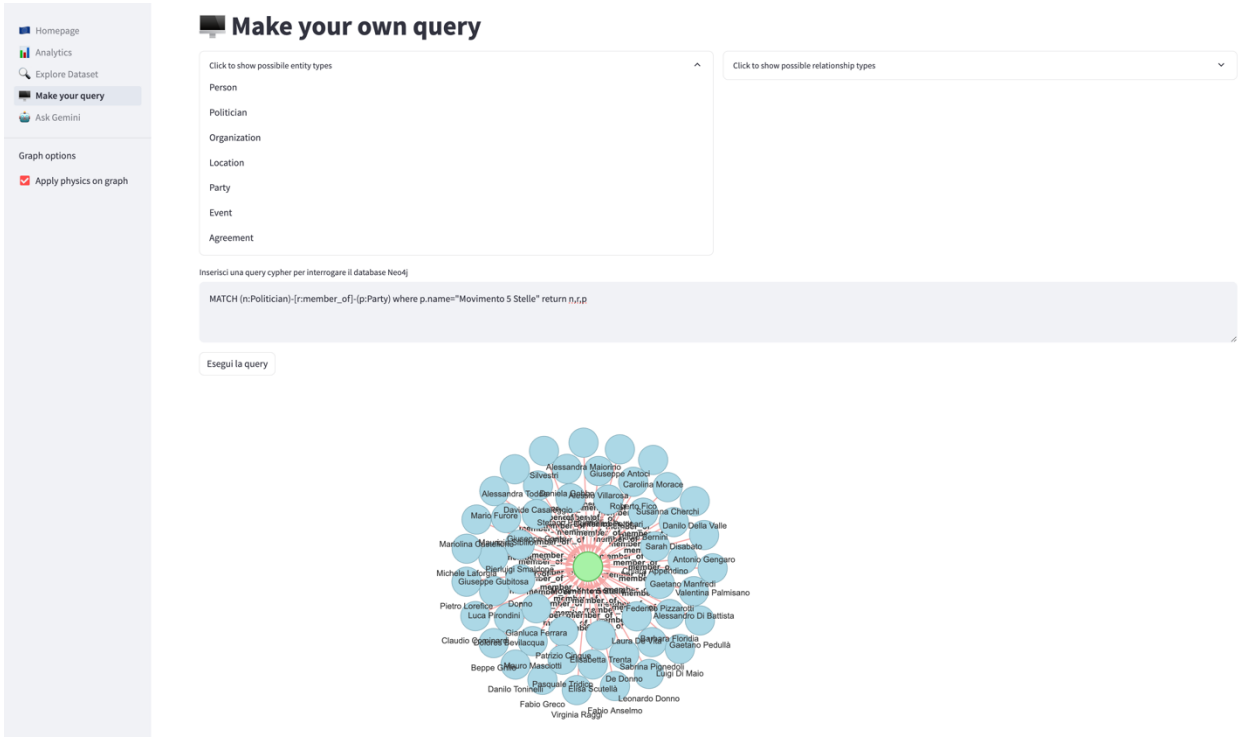


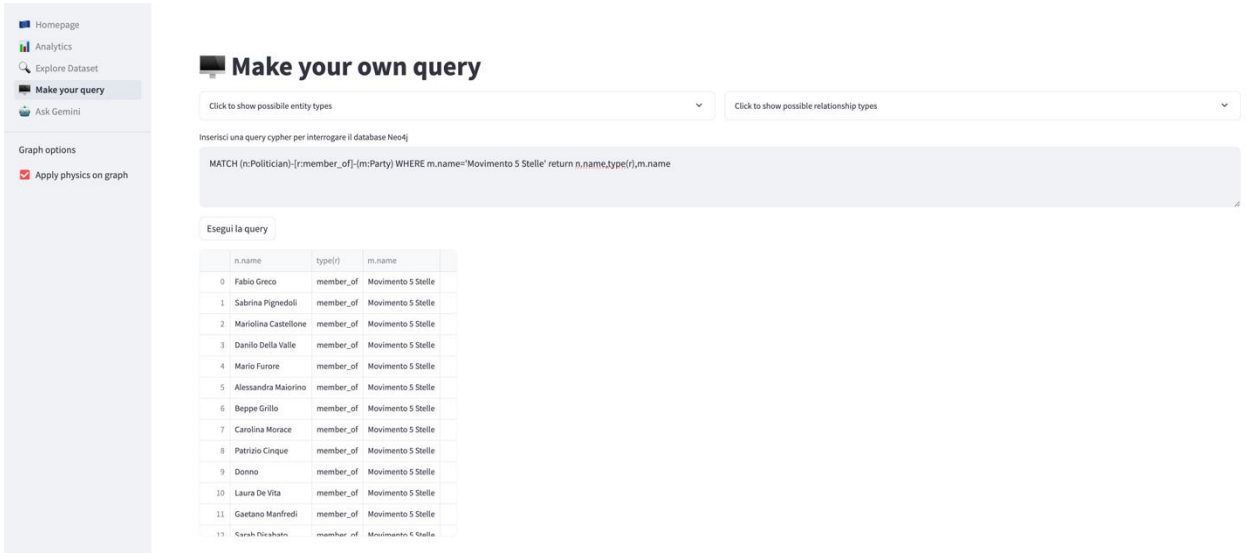


Nella pagina Explore Dataset, abbiamo inserito delle checkbox relative ai tipi delle entità e delle relazioni presenti nel nostro database, spuntando ciasuna checkbox viene eseguita una query al DB e mostrati a video i risultati di quest'ultima sotto forma di dataframe.



Nella pagina Make your query viene messa a disposizione dell'utente la possibilità di eseguire una query a piacere in linguaggio cypher. Questa query viene inviata al DB tramite il seguente URI: bolt://localhost:7687.
L'output della query viene mostrato sulla pagina o sotto forma di grafo o sotto forma di dataframe a seconda della tipologia di query eseguita, ad esempio se nella query vengono ritornati solo nodi e relazioni, l'output sarà un grafo, nel caso in cui invece viene ritornata almeno una proprietà, l'output sarà un dataframe.
È possibile, inoltre, scegliere se applicare o meno la fisica al grafo.





L’ultima pagina: Ask Gemini, permette di interrogare il modello LLM Gemini, al quale è stato dato il compito di: “a partire da una domanda in linguaggio naturale, generare query in linguaggio Cypher”.
L’utente inserisce una richiesta nella prima text box, e dopo aver premuto il button “Generate query”, questa viene inviata al modello, la cui risposta verrà inserita nella seconda text box.
A questo punto l’utente ha la possibilità di eseguire direttamente la query tramite il button “Execute query” e vederne i risultati.