

Progetto Finale - Big Data Engineering



UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

CORSO DI BIG DATA ENGINEERING - LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ANNO ACCADEMICO 2022-2023

PROGETTO DI HEALTHCARE SUMMARIZATION

Big Data Engineering

Healthcare Summarization di MIMIC-III

Concept Extraction e Summarization tramite NER e Generative Language Model

Professore:

Ing. Vincenzo Moscato

Assistant:

Ing. Marco Postiglione

Studenti:

Antonio Romano M63001315

Andriy Korsun M63001275

Giuseppe Riccio M63001314

Michele Cirillo M63001293

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 1 |
| 1.1 | Contesto del progetto | 1 |
| 1.1.1 | Scopo del progetto | 1 |
| 1.2 | Sorgente dati | 1 |
| 1.2.1 | Accesso ai dati del MIMIC-III | 3 |
| 1.3 | Workflow del progetto | 4 |
| 2 | Data Extraction | 5 |
| 2.1 | Recupero dei dati Target dai dati Raw | 5 |
| 2.1.1 | Note Events in MIMIC-III | 5 |
| 2.1.1.1 | Estrazione dei dati dal Note Events | 5 |
| 2.1.1.2 | Distribuzione dei documenti per categoria | 6 |
| 2.1.1.3 | Distribuzione dei documenti basata sulla loro lunghezza | 7 |
| 2.1.1.4 | Numero di documenti per paziente | 7 |
| 2.1.1.5 | Estrazione delle colonne d'interesse da Note Events | 8 |
| 2.1.1.6 | Caratterizzazione dei documenti: Token Count | 10 |
| 2.1.1.7 | Analisi e filtraggio dei documenti | 10 |
| 2.1.1.8 | Distribuzione dei documenti basata sulla loro lunghezza - Dataframe estratto | 11 |
| 2.1.1.9 | Numero di documenti per paziente - Dataframe estratto | 12 |
| 2.1.1.10 | Duplicate Row and Token.Count Remove | 12 |
| 2.1.2 | Patients in MIMIC-III | 13 |
| 2.1.2.1 | Estrazione dei dati da Patients | 13 |
| 2.1.2.2 | Estrazione delle colonne d'interesse da Patients | 13 |
| 2.1.3 | Dataframe finale estratto da NOTEVENTS e PATIENTS | 14 |
| 3 | Memorizzazione dei dati | 15 |
| 3.1 | Tecnologie usate | 15 |
| 3.1.1 | MongoDB | 15 |
| 3.1.1.1 | MongoDB Atlas | 16 |
| 3.1.1.2 | Workflow: Data Collection → Python → MongoDB Atlas | 16 |
| 3.1.2 | Neo4J | 17 |
| 3.2 | Architettura finale di memorizzazione Big Data | 18 |
| 3.2.1 | Analisi di scalabilità | 19 |
| 4 | Data preprocessing | 20 |
| 4.1 | Selezione dei pazienti | 20 |
| 4.1.1 | Implementazione | 20 |
| 4.2 | Section Extraction | 22 |
| 4.2.1 | Implementazione | 22 |
| 4.3 | Stopwords and Lemmatization | 23 |
| 4.3.1 | Implementazione | 23 |
| 4.4 | Pulizia delle note cliniche | 24 |
| 4.4.1 | Implementazione | 24 |

| | |
|---|-----------|
| 5 Generative Language Model | 26 |
| 5.1 Prompt | 27 |
| 5.2 Summary | 27 |
| 5.2.0.1 Prompt | 27 |
| 5.2.1 Il Modello | 28 |
| 5.3 Clinical Trend | 29 |
| 5.3.0.1 Prompt | 29 |
| 5.3.1 Il Modello | 30 |
| 5.4 Salvataggio dei Summary e dei Clinical Trend | 31 |
| 6 Named Entity Recognition | 32 |
| 6.1 MedCAT | 32 |
| 6.1.1 Implementazione | 33 |
| 6.2 Relation Extraction | 41 |
| 6.2.1 Prompt - Relation Extraction | 41 |
| 6.2.2 Implementazione | 42 |
| 7 Graph Modeling | 46 |
| 7.1 Memorizzazione su Neo4J | 46 |
| 7.2 Concept Graph | 47 |
| 7.2.1 Schema | 47 |
| 7.2.2 Network esteso | 48 |
| 7.3 Knowledge Graph | 49 |
| 7.3.1 Schema | 49 |
| 7.3.2 Network esteso | 49 |
| 7.3.3 Entità collegate alla nota clinica | 50 |
| 7.3.4 Relazioni per entità cliniche | 52 |
| 7.4 Risultati | 53 |
| 8 Analytics and Report | 55 |
| 8.1 Dashboard | 55 |
| 8.2 Analytics | 57 |
| 8.2.1 Informazioni demografiche e cliniche del paziente | 57 |
| 8.2.1.1 Implementazione | 57 |
| 8.2.1.2 Risultati | 57 |
| 8.2.2 Estrazione Clinical Trend | 59 |
| 8.2.2.1 Implementazione | 59 |
| 8.2.2.2 Risultati | 59 |
| 8.2.3 Estrazione Summary | 60 |
| 8.2.3.1 Implementazione | 60 |
| 8.2.3.2 Risultati | 60 |
| 8.2.4 Numero di malattie/sintomi diagnosticati nel tempo | 61 |
| 8.2.4.1 Implementazione | 61 |
| 8.2.4.2 Risultati | 61 |
| 8.2.5 Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo | 63 |
| 8.2.5.1 Implementazione | 63 |
| 8.2.5.2 Risultati | 63 |
| 8.2.6 Elenco di farmaci, sintomi e diagnostiche | 65 |
| 8.2.6.1 Implementazione | 65 |
| 8.2.6.2 Risultati | 65 |

| | |
|---|-----------|
| 8.2.7 Patologie del paziente associate a note cliniche | 67 |
| 8.2.7.1 Implementazione | 67 |
| 8.2.7.2 Risultati | 67 |
| 8.2.8 Parti del corpo doloranti del paziente associate alle note cliniche | 68 |
| 8.2.8.1 Implementazione | 68 |
| 8.2.8.2 Risultati | 68 |
| 8.2.9 Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche | 69 |
| 8.2.9.1 Implementazione | 69 |
| 8.2.9.2 Risultati | 69 |
| 8.2.10 Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche | 70 |
| 8.2.10.1 Implementazione | 70 |
| 8.2.10.2 Risultati | 70 |
| 8.2.11 Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche | 71 |
| 8.2.11.1 Implementazione | 71 |
| 8.2.11.2 Risultati | 71 |
| 8.2.12 Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche | 72 |
| 8.2.12.1 Implementazione | 72 |
| 8.2.12.2 Risultati | 72 |
| 9 Conclusioni | 73 |
| 9.1 Efficienza e riduzione del carico di lavoro | 73 |
| 9.2 Miglioramento della qualità dell'assistenza | 73 |
| 9.3 Supporto decisionale clinico | 73 |
| 9.4 Costi di realizzazione del progetto effettivi e futuri | 73 |
| 9.5 Tempi di esecuzione | 75 |
| 9.6 Sviluppi futuri: Migrazione da MongoDB e Neo4J verso ArangoDB | 76 |
| 9.7 Materiale del progetto | 76 |
| Elenco delle figure | 77 |
| Elenco delle tabelle | 78 |
| Bibliografia | 79 |

1 Introduzione

Contenuti

| | | |
|-------|-------------------------------|---|
| 1.1 | Contesto del progetto | 1 |
| 1.1.1 | Scopo del progetto | 1 |
| 1.2 | Sorgente dati | 1 |
| 1.2.1 | Accesso ai dati del MIMIC-III | 3 |
| 1.3 | Workflow del progetto | 4 |

1.1 Contesto del progetto

Le tecniche di sintesi del testo sono diventate cruciali per affrontare la grossa quantità di informazioni testuali disponibili sul web. Queste tecniche consentono di estrarre in modo efficace informazioni rilevanti da grandi collezioni di documenti per poi fornire rappresentazioni concise dei documenti preservando i dettagli più importanti. Tuttavia, riassumere documenti come articoli di notizie, scientifici o post sui social media risulta spesso complicato a causa delle loro strutture e dipendenze. Per creare riassunti che catturino i temi principali e le idee espresse lungo la sequenza mantenendo coerenza e contesto, sono necessari nuovi approcci.

L'utilizzo di un **prompt** in input ad un modello GLM (Generative Language Model) è emerso come una tecnica utile nel campo della sintesi, particolarmente per documenti che descrivono l'evoluzione di eventi o delle entità nel tempo. Le indicazioni fornite nei prompt appaiono come delle frasi guida o domande che indirizzano il processo di sintesi verso obiettivi specifici. Sfruttando le indicazioni, i sistemi di sintesi possono generare riassunti che più si adattano alle esigenze specifiche di un dominio di interesse. Questo approccio è reso possibile dallo sviluppo dei sistemi di intelligenza artificiale generativa, che hanno migliorato l'efficacia e l'efficienza di tali tecniche di sintesi.

Nel caso in questione, questo processo è inteso essere applicato nel dominio della **Healthcare**, specificamente a partire da un dataset di registri clinici appartenenti al database MIMIC-III.

1.1.1 Scopo del progetto

L'obiettivo di questo progetto è quello di **estrarre** tutti i dati testuali semi-strutturati scritti da medici o infermieri durante la storia medica di un paziente dal database MIMIC-III (**noteevents**). Questi dati testuali verranno successivamente analizzati per generare un rapporto completo che riassume la storia medica di un paziente. Il rapporto includerà informazioni come condizioni mediche, farmaci prescritti e eventuali procedure rilevanti eseguite.

1.2 Sorgente dati

Il dato in ingresso del progetto d'esame è MIMIC. **MIMIC-III (Medical Information Mart for Intensive Care III)** [7] è un database relazionale pubblico contenente dati sanitari de-identificati relativi a pazienti in cure intensive. È un database ampiamente utilizzato nella comunità della ricerca sanitaria ed è gestito dal Massachusetts Institute of Technology (MIT). MIMIC-III include dati clinici completi raccolti dalle unità di terapia intensiva (ICU) del Beth Israel Deaconess Medical Center di Boston, Massachusetts, nel periodo dal 2001 al 2012 relativo a 47,591 ricoveri.

I dati clinici del database MIMIC III sono anonimizzati

Il **database** contiene informazioni come dati demografici, segni vitali, misurazioni di laboratorio, farmaci, procedere, note del personale sanitario e altro ancora. Fornisce ai ricercatori una preziosa risorsa per studiare vari aspetti delle cure intensive, condurre analisi retrospettive, sviluppare modelli predittivi e valutare strategie di trattamento.

MIMIC-III è un database relazionale composto da 26 tabelle. Le tabelle sono collegate tramite identificatori che di solito hanno il suffisso 'ID'.

Ecco un elenco delle principali **tabelle disponibili**¹ nel database MIMIC-III:

● **Tabelle principali:**

- **PATIENTS:** Informazioni demografiche dei pazienti, come età, sesso ed etnia;
- **ADMISSIONS:** Dettagli sui ricoveri dei pazienti, compresi i tempi di ricovero e dimissione;
- **ICUSTAYS:** Informazioni sui soggiorni in unità di terapia intensiva (ICU), come unità di terapia intensiva, date di ricovero e dimissione;
- **DIAGNOSES_ICD:** Codici diagnostici ICD-9 relativi alle diagnosi dei pazienti;
- **PROCEDURES_ICD:** Codici ICD-9 relativi alle procedure eseguite sui pazienti.

● **Tabelle dei segni vitali:**

- **CHARTEVENTS:** Registrazioni del monitoraggio dei segni vitali, valori di laboratorio e segni neurologici;
- **LABEVENTS:** Risultati dei test di laboratorio condotti sui pazienti;
- **OUTPUТЕVENTS:** Informazioni sull'ingresso e l'uscita di fluidi somministrati ai pazienti.

● **Note e tabelle di testo:**

- **NOTEVENTS:** Note del medico e dell'infermiere relative ai pazienti;
- **PRESCRIPTIONS:** Informazioni sulle prescrizioni di farmaci per i pazienti.

● **Tabelle specifiche per terapia intensiva ICU:**

- **D_ITEMS:** Informazioni sugli elementi di misurazione e sui dispositivi utilizzati in terapia intensiva;
- **INPUTEVENTS:** Dettagli sugli input di farmaci e fluidi somministrati ai pazienti in terapia intensiva;
- **PROCEDUREEVENTS:** Informazioni sulle procedure mediche eseguite su pazienti in terapia intensiva;
- **MICROLOGYEVENTS:** Risultati dei test microbiologici condotti su pazienti in terapia intensiva.

● **Tabelle ausiliarie:**

- **D_ICD_DIAGNOSES:** Descrizioni testuali corrispondenti ai codici diagnostici ICD-9;
- **D_ICD PROCEDURES:** Descrizioni testuali corrispondenti ai codici di procedura ICD-9;
- **D_CPT:** Informazioni sui codici di procedura corrispondenti alla Current Procedural Terminology (CPT);
- **D_ITEMS:** Descrizioni testuali degli elementi di misurazione e dei dispositivi utilizzati in terapia intensiva.

¹<https://mimic.mit.edu/docs/iii/tables/>

1.2.1 Accesso ai dati del MIMIC-III

Per accedere al database MIMIC-III, occorre seguire i passaggi descritti di seguito:

1. **Ottenere l'accesso:** MIMIC-III è un database pubblicamente disponibile, tuttavia l'accesso deve essere richiesto. Attraverso il sito Web ufficiale di MIMIC-III si accede alla sezione "Accesso a MIMIC" ². Lì sono contenute informazioni dettagliate su come ottenere l'accesso e i requisiti necessari per garantire un uso etico;
2. **Completare la formazione necessaria:** come parte del processo di accesso, è richiesto di completare moduli di formazione ³ specifici relativi alla riservatezza dei dati, alla sicurezza e alla gestione delle informazioni sensibili del paziente. Questa formazione è progettata per garantire la comprensione delle responsabilità etiche e legali associate all'utilizzo del set di dati;
3. **Firma l'accordo sull'utilizzo dei dati:** dopo aver completato la formazione richiesta, ti verrà chiesto di firmare un accordo sull'utilizzo dei dati ⁴. Questo accordo delinea i termini e le condizioni per l'utilizzo del database MIMIC-III, inclusa la riservatezza dei dati, la riservatezza e l'uso appropriato;
4. **Scarica il database:** una volta approvata la tua richiesta di accesso e completati tutti i passaggi necessari, riceverai le istruzioni su come scaricare il database MIMIC-III. Il set di dati è disponibile in un formato strutturato, come i file CSV, e potrebbe richiedere uno spazio di archiviazione significativo a causa delle sue dimensioni.

È essenziale notare che MIMIC-III contiene dati sensibili del paziente e, pertanto, deve essere gestito con la massima cura e nel rispetto delle normative sulla privacy e delle politiche istituzionali. Prima di utilizzare il set di dati, rivedere attentamente e aderire alle linee guida fornite dal team MIMIC e consultare il comitato etico locale.

ATTENZIONE: Nel caso in esame, per ottenere l'accesso a questi dati è stato necessario seguire i seguenti passaggi. Inoltre, si ricorda che i dati non possono essere condivisi pubblicamente nella loro interezza e per questo motivo nel prosieguo del documento saranno mostrati solo estratti di essi.

²<https://physionet.org/content/mimiciii/1.4/>

³<https://physionet.org/about/citi-course/>

⁴<https://physionet.org/login/?next=/settings/credentialing/>

1.3 Workflow del progetto

Nella seguente Figura 1.1 si mostra il workflow completo seguito nel progetto:

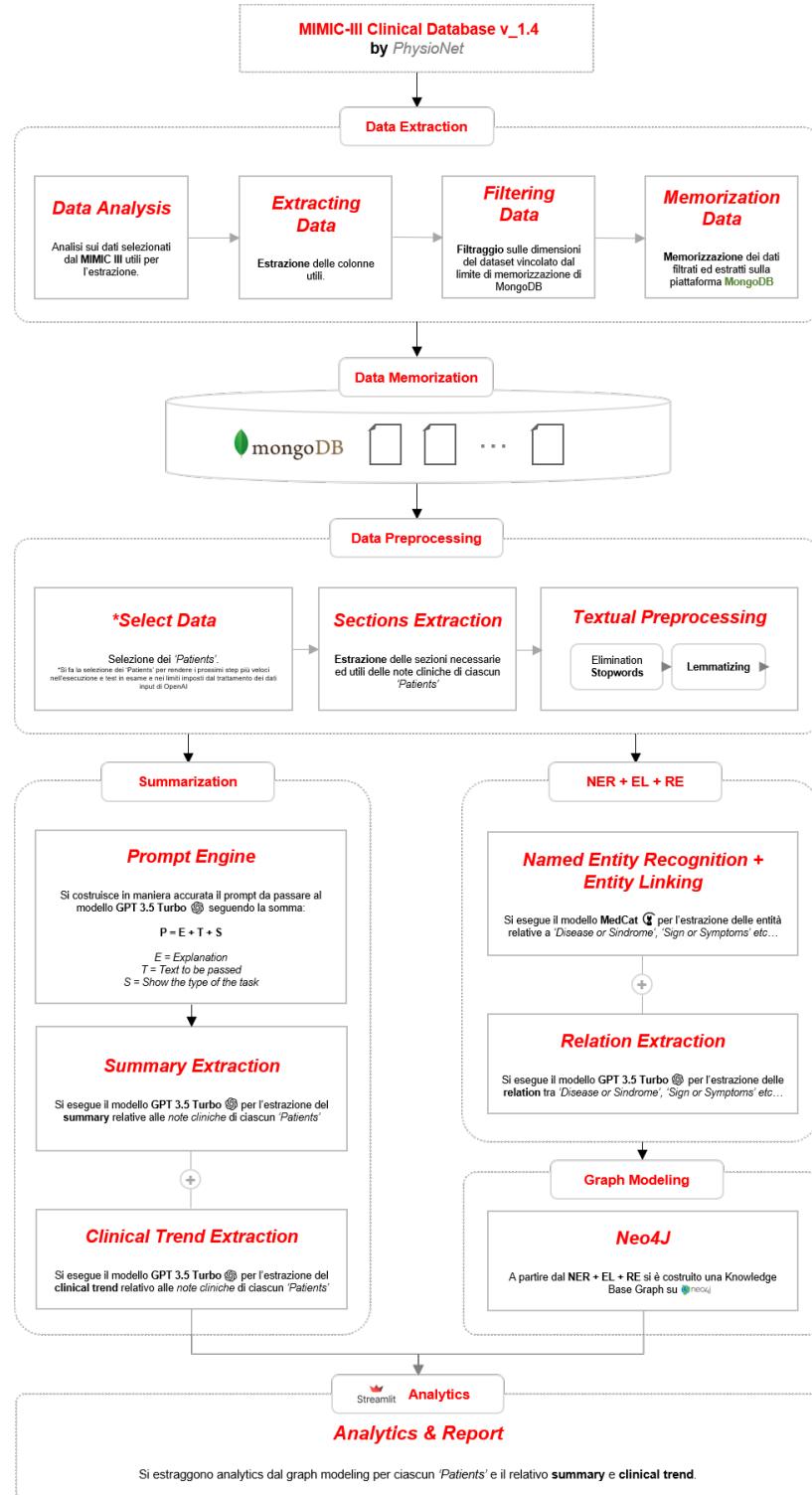


Figura 1.1: Workflow del progetto

2 Data Extraction

Contenuti

| | |
|---|----|
| 2.1 Recupero dei dati Target dai dati Raw | 5 |
| 2.1.1 Note Events in MIMIC-III | 5 |
| 2.1.2 Patients in MIMIC-III | 13 |
| 2.1.3 Dataframe finale estratto da NOTEVENTS e PATIENTS | 14 |

2.1 Recupero dei dati Target dai dati Raw

Questa sezione del Capitolo 2 si concentra sulla lettura, analisi ed estrazione dei dati a partire dalle annotazioni mediche contenute nella collezione "NOTEVENTS" all'interno del database MIMIC-III.

2.1.1 Note Events in MIMIC-III

Note Events in MIMIC-III fa riferimento ad una collezione di annotazioni testuali, annotate dai professionisti sanitari durante la cura dei pazienti, che includono note di ammissione e dimissione, prescrizioni mediche, referti medici ed altro. Queste annotazioni forniscono dettagli importanti anche sulla storia medica di un paziente, come diagnosi cliniche, procedure eseguite, trattamenti prescritti, risultati dei test di laboratorio.

2.1.1.1 Estrazione dei dati dal Note Events

L'estrazione dei dati, a partire da NOTEVENTS.csv, è stata effettuata con il seguente comando:

```
df = pd.read_csv('NOTEVENTS.csv', header=0)
```

Il dataframe ottenuto consta di 2083180 righe e di 11 colonne, ovvero:

- **ROW_ID**: identificatore unico per ogni riga;
- **SUBJECT_ID**: identificatore univoco associato a ciascun paziente;
- **HADM_ID**: identificatore univoco associato a ciascun episodio di ammissione ospedaliera;
- **CHARTDATE**: data in cui è stata generata l'annotazione medica;
- **CHARTTIME**: ora in cui è stata generata l'annotazione medica;
- **STORETIME**: ora in cui l'annotazione medica è stata memorizzata;
- **CATEGORY**: categoria o tipo di annotazione medica;
- **DESCRIPTION**: descrizione o titolo più dettagliato della categoria dell'annotazione medica;
- **CGID**: identificatore univoco associato all'utente o al gruppo che ha generato l'annotazione medica;
- **ISERROR**: indicatore che segnala se ci sono errori o problemi con l'annotazione medica;

- **TEXT:** testo completo dell'annotazione medica, che contiene le informazioni rilevanti sulla storia medica del paziente o altri dettagli clinici.

Seguiranno, nelle sezioni successive, delle analisi su di esso.

2.1.1.2 Distribuzione dei documenti per categoria

Verrà mostrata di seguito l'implementazione ed il risultato in uscita per l'analisi della distribuzione dei documenti per categoria, utile per sapere da dove provengono i documenti:

```
plot = sns.countplot(x=df['CATEGORY'],
                     order=df['CATEGORY'].value_counts().index)
plot.set_xticklabels(plot.get_xticklabels(),
                     rotation=45, horizontalalignment='right')
plt.xlabel('Categoria')
plt.ylabel('Numero Documenti')
```

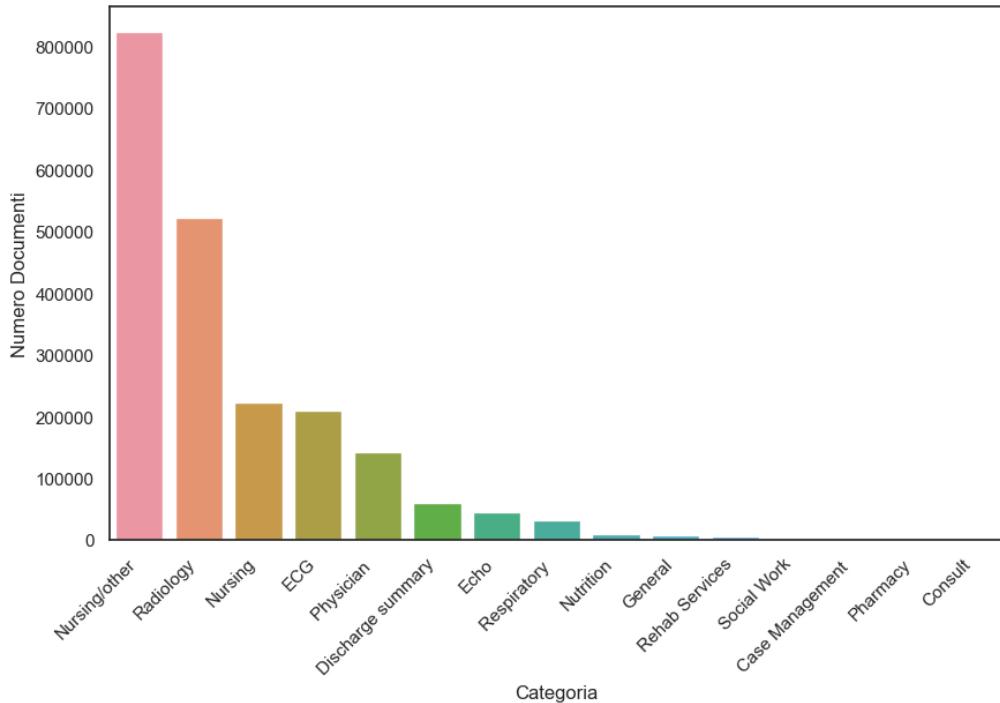


Figura 2.1: Plot - Distribuzione Documenti per CATEGORIA

Come si può notare dalla precedente Figura, la categoria di documenti più frequente è quella relativa alle note scritte dagli infermieri. Tuttavia, la categoria che ci interessa di più è quella dei "**Discharge summary**"; infatti, tali note **contengono informazioni complete** relative all'intero periodo di ricovero del paziente in ospedale.

2.1.1.3 Distribuzione dei documenti basata sulla loro lunghezza

Verrà mostrata di seguito l'implementazione ed il risultato in uscita per l'analisi della distribuzione dei documenti in base alla lunghezza (numero di caratteri) delle note cliniche:

```
lns = [len(str(x)) for x in df['TEXT']]  
sns.histplot(lns, kde=False)  
plt.xlabel('Lunghezza Documenti')  
plt.ylabel('Numero Documenti')
```

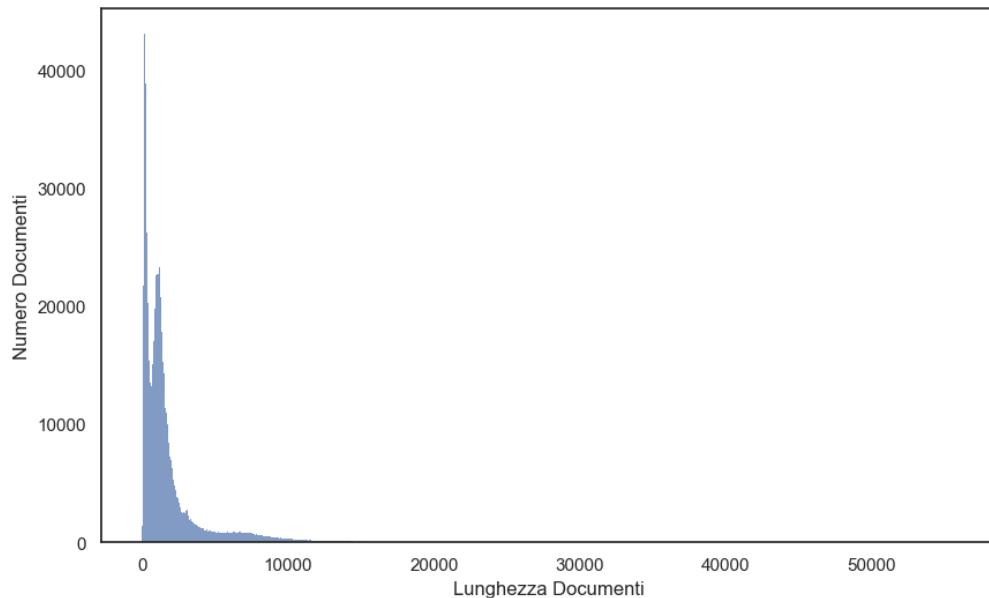


Figura 2.2: Plot - Distribuzione Documenti per Lunghezza Carattere

Nella Figura precedente, si può notare come la lunghezza massima dei documenti si aggira intorno ai 10.000 caratteri, con una maggior concentrazione intorno all'intervallo che va dai 100 ai 5.000 caratteri.

2.1.1.4 Numero di documenti per paziente

Verrà mostrata di seguito l'implementazione ed il risultato in uscita per l'analisi del numero di documenti per paziente:

```
sns.histplot(df['SUBJECT_ID'].value_counts().values,  
             kde=False)  
plt.xlabel('Documenti per Paziente')  
plt.ylabel('Numero Documenti')
```

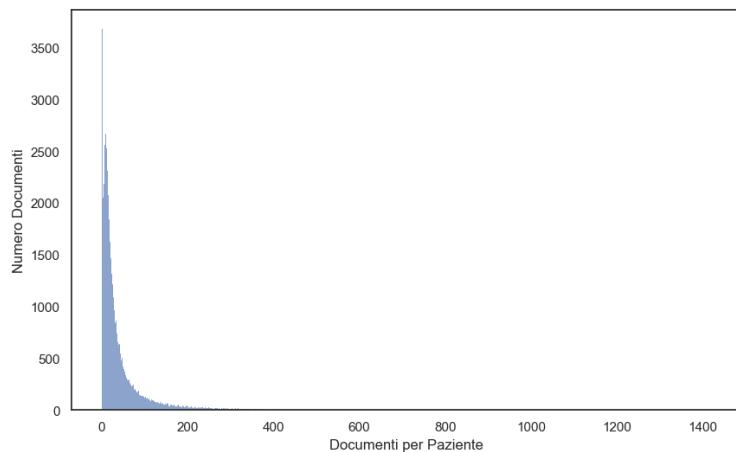


Figura 2.3: Plot - Numero Documenti per Paziente

Dalla Figura precedente, si evince che la maggior parte dei pazienti ha un numero di documenti che si aggira nel range tra 1 e 100 documenti.

Si noti: queste distribuzioni saranno utili per i prossimi paragrafi in quanto sarà necessario filtrare il numero delle righe del dataset noteevents in funzione dei limiti imposti sia delle dimensioni totali che mette a disposizione MongoDB, sia della lunghezza massima dei documenti in input al prompt del modello GLM.

2.1.1.5 Estrazione delle colonne d'interesse da Note Events

Precedentemente, si è descritto il dataframe ottenuto dall'estrazione delle informazioni contenute all'interno di NOTEVENTS.csv. Più nello specifico, lo si è descritto come un dataframe contenente 2083180 righe ed 11 colonne. Tuttavia, non tutte le informazioni sono necessarie per l'estrazione dei **concetti medici** (che si vedrà nella fase di **NER**). Proprio per questo, si è deciso di rimuovere colonne non utili ai fini delle fasi successive. In particolare, con l'implementazione proposta si trascurano le seguenti colonne:

- **HADM_ID**;
- **CHARTTIME**;
- **STORETIME**;
- **DESCRIPTION**;
- **CGID**.

```
filtered_df = df.drop('HADM_ID', axis=1)
filtered_df = filtered_df.drop('CHARTTIME', axis=1)
filtered_df = filtered_df.drop('STORETIME', axis=1)
filtered_df = filtered_df.drop('DESCRIPTION', axis=1)
filtered_df = filtered_df.drop('CGID', axis=1)
```

Si vuole ricordare che la colonna **ISERROR** è un indicatore che segnala se ci sono errori o problemi con l'annotazione medica.

Ai fini del progetto, si è scelto di **trascurare le righe che presentano annotazioni errate**, con problemi, andando quindi a filtrare le righe in base alla condizione che il valore nella colonna "ISERROR" sia diverso da 1, mantenendo solo le righe che soddisfano tale condizione:

```
filtered_df = filtered_df[filtered_df['ISERROR'] != 1]
```

Si ricorda, inoltre, che la colonna **CATEGORY** rappresenta la categoria o tipo di annotazione medica.

Vengono filtrate ulteriormente le righe in base alla condizione che il valore nella colonna "CATEGORY" sia uguale a "Discharge summary", mantenendo solo le righe che soddisfano tale condizione:

```
filtered_df = filtered_df[filtered_df['CATEGORY'] ==  
    "Discharge summary"]
```

Con "Discharge Summary" facciamo riferimento alle annotazioni mediche che contengono riassunti delle informazioni cliniche e delle procedure relative alla dimissione di un paziente dall'ospedale. Queste annotazioni vengono create quando un **paziente viene dimesso** e forniscono un riepilogo delle condizioni di salute del paziente durante il periodo di ricovero, i trattamenti e le terapie somministrate, i farmaci prescritti e altre informazioni rilevanti per la continuità delle cure o il monitoraggio successivo del paziente.

Ed infine, si procede con la rimozione di esse dal dataframe:

```
filtered_df = filtered_df.drop('CATEGORY', axis=1)  
filtered_df = filtered_df.drop('ISERROR', axis=1)
```

Ottenendo un dataframe filtrato di **59652** righe e **4** colonne:

| ROW_ID | SUBJECT_ID | CHARTDATE | TEXT |
|--------|------------|------------|--|
| 174 | 22532 | 2151-08-04 | Admission Date: [**2151-7-16**] Dischar... |
| 175 | 13702 | 2118-06-14 | Admission Date: [**2118-6-2**] Discharg... |
| 176 | 13702 | 2119-05-25 | Admission Date: [**2119-5-4**] D... |
| 177 | 13702 | 2124-08-18 | Admission Date: [**2124-7-21**] ... |
| 178 | 26880 | 2162-03-25 | Admission Date: [**2162-3-3**] D... |
| ... | ... | ... | ... |

Tabella 2.1: Dataframe estratto da Note Events

2.1.1.6 Caratterizzazione dei documenti: Token Count

In questa fase di progetto, si analizzano i documenti per ogni paziente e per ogni cartella clinica ad essi associati. Si esegue questa operazione in vista dei limiti imposti dal modello GPT come detto in precedenza.

Pertanto, una prima operazione effettuata è quella di visionare, in maniera tabellare, la dimensione effettiva dei documenti andando ad effettuare un conteggio dei **Token** presenti nel campo "TEXT".

Non esiste una misura precisa del conteggio dei token attraverso il conteggio delle words, ma generalmente:

- 1 token = 4 chars in English;
- 1 token = $\frac{3}{4}$ words;
- 100 tokens = 75 words.

Attraverso la seguente implementazione:

```
filtered_df['TOKEN_COUNT'] = filtered_df['TEXT'].str.len()
/ 4
```

è stato possibile **stimare** il numero di Token presenti nella colonna Text, ed è stata in seguito aggiunta come colonna nel dataframe. Il risultato lo si mostra in tabella sottostante:

| ROW_ID | SUBJECT_ID | CHARTDATE | TEXT | TOKEN_COUNT |
|--------|------------|------------|--|-------------|
| 174 | 22532 | 2151-08-04 | Admission Date: [**2151-7-16**] Dischar... | 222 |
| 175 | 13702 | 2118-06-14 | Admission Date: [**2118-6-2**] Discharg... | 3158 |
| 176 | 13702 | 2119-05-25 | Admission Date: [**2119-5-4**] D... | 2434 |
| 177 | 13702 | 2124-08-18 | Admission Date: [**2124-7-21**] ... | 4284 |
| 178 | 26880 | 2162-03-25 | Admission Date: [**2162-3-3**] D... | 3625 |
| ... | ... | ... | ... | ... |

Tabella 2.2: Dataframe estratto da Note Events con Token Count

Grazie a questo conteggio sarà possibile supporre il numero massimo di token totali tenendo conto dei possibili processi di eliminazione delle stopwords e lemmatizing. Quest'ultime fasi, in media riescono ad eliminare 1/3 del testo. Si è pertanto limitato la somma totale dei token delle note complessive di ciascun paziente intorno ai 10000.

2.1.1.7 Analisi e filtraggio dei documenti

Dopo aver calcolato il numero di Token per ogni campo "TEXT" presente nei documenti dei pazienti, occorre valutare quale di essi occorre mantenere ai fini dell'analisi.

A favore di ciò, diverse operazioni sono state effettuate sul dataframe contenente i documenti dei pazienti:

- vengono mantenuti documenti dei pazienti (raggruppati con il campo SUBJECT_ID) che presentano una somma dei valori nella colonna "**TOKEN_COUNT**" **inferiore o uguale a 10000**. Questo perché trattare documenti molto corposi significa avere lentezza nella loro analisi automatizzata, non avendo a disposizione architetture dedicate di Big Data le quali hanno costi significativi;

- vengono mantenuti i documenti dei pazienti che presentano, come valore del campo **"TOKEN_COUNT"**, **un numero maggiore o uguale a 500**. Ciò perché avere pochi dati in un documento potrebbe portare ad analisi poco chiare a causa di un numero di informazioni insufficienti;
- vengono mantenuti i documenti dei pazienti, attraverso il campo **"SUBJECT_ID"** che presentano almeno 2 documenti ad essi associati. Ciò aiuta ad effettuare analisi più precise rispetto all'avere un solo documento essendo anche richiesta un summary della loro storia medica.

Queste tre operazioni, possono essere schematizzate attraverso la seguente implementazione:

```
filtered_df = filtered_df.groupby('SUBJECT_ID')
    .filter(lambda group: group['TOKEN_COUNT'].sum() <= 10000)
filtered_df = filtered_df[filtered_df['TOKEN_COUNT'] >= 500]
filtered_df = filtered_df.groupby('SUBJECT_ID')
    .filter(lambda x: len(x) > 1)
```

2.1.1.8 Distribuzione dei documenti basata sulla loro lunghezza - Dataframe estratto

Verrà mostrata di seguito l'implementazione ed il risultato in uscita per l'analisi della distribuzione dei documenti in base alla lunghezza (numero di caratteri) delle note cliniche, questa volta applicata sul dataframe estratto (discussa precedentemente):

```
lns = [len(str(x)) for x in filtered_df['TEXT']]
sns.histplot(lns, kde=False)
plt.xlabel('Lunghezza Documenti')
plt.ylabel('Numero Documenti')
```

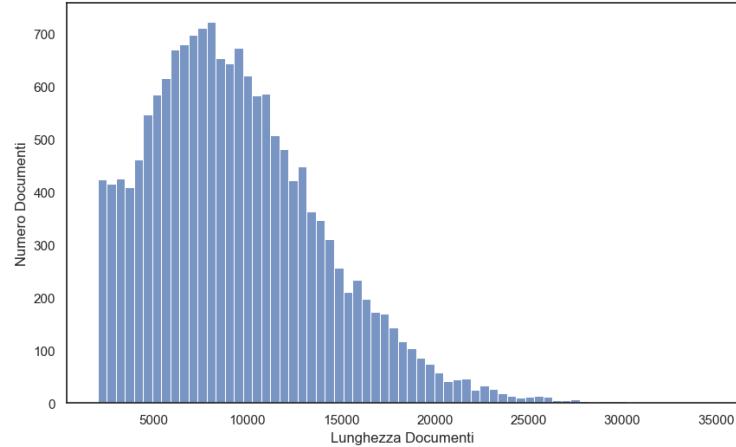


Figura 2.4: Plot - Distribuzione Documenti per Lunghezza Carattere

2.1.1.9 Numero di documenti per paziente - Dataframe estratto

Verrà mostrata di seguito l'implementazione ed il risultato in uscita per l'analisi del numero di documenti per paziente, questa volta applicata sul dataframe estratto (discussa precedentemente):

```
sns.histplot(filtered_df['SUBJECT_ID'].value_counts().values,  
             kde=False)  
plt.xlabel('Documenti per Paziente')  
plt.ylabel('Numero Documenti')
```

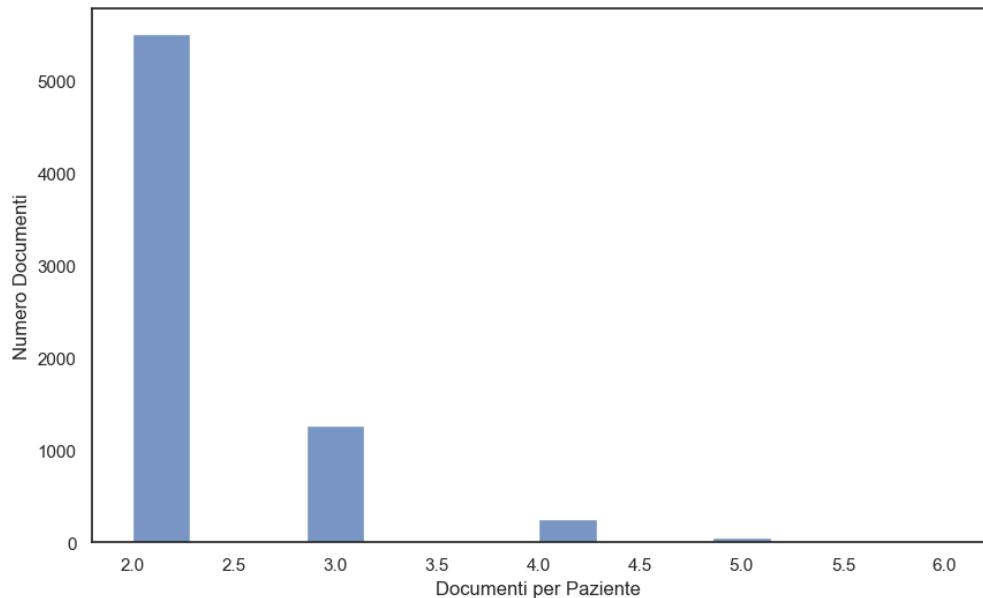


Figura 2.5: Plot - Numero Documenti per Paziente

2.1.1.10 Duplicate Row and Token.Count Remove

Come ultima operazione, vengono rimosse, con il seguente script, le righe duplicate nel dataframe filtrato, basandosi sulla colonna "TEXT". Ciò significa che se due righe hanno lo stesso valore nella colonna "TEXT", verrà mantenuta solo una delle due righe e le altre verranno eliminate. Inoltre, viene eliminata la colonna "TOKEN_COUNT" in quanto non utile ai fini delle successive analisi:

```
filtered_df.drop_duplicates(subset='TEXT', inplace=True)  
filtered_df = filtered_df.drop('TOKEN_COUNT', axis=1)
```

Ottenendo il dataframe filtrato finale che consta di **16120** righe e **4** colonne.

2.1.2 Patients in MIMIC-III

Patients in MIMIC-III fa riferimento ad una collezione di informazioni demografiche riguardanti i pazienti dell'ospedale, quali ad esempio, genere, data di nascita e data di morte.

2.1.2.1 Estrazione dei dati da Patients

L'estrazione dei dati, a partire da PATIENTS.csv, è stata effettuata con il seguente comando:

```
patient_df = pd.read_csv('PATIENTS.csv', header=0)
```

Il dataframe ottenuto consta di 46520 righe e di 8 colonne, ovvero:

- **ROW_ID**: identificatore unico per ogni riga;
- **SUBJECT_ID**: identificatore univoco associato a ciascun paziente;
- **GENDER**: sesso genotipico del paziente;
- **DOB**: data di nascita del paziente;
- **DOD**: data di morte per il paziente;
- **DOD_HOSP**: data del decesso registrata nel database dell'ospedale;
- **DOD_SS**: data del decesso dal database della previdenza sociale;
- **EXPIRE_FLAG**: flag binario che indica se il paziente è deceduto o meno.

Seguiranno, nelle sezioni successive, delle operazioni su di esso.

2.1.2.2 Estrazione delle colonne d'interesse da Patients

Precedentemente, abbiamo estratto il dataframe delle Note Events al cui interno sono presenti delle informazioni sulle note cliniche del paziente. Tuttavia, non abbiamo notizie sul paziente stesso; per questo motivo, andremo ad unire al dataframe precedente i dati dei pazienti estratti a partire da PATIENTS.csv.

Tuttavia, non tutte le informazioni sono necessarie per eseguire il merge tra questi dataframe come detto in precedenza. Proprio per questo, si è deciso di rimuovere colonne non utili ai fini delle fasi successive. In particolare, con l'implementazione proposta si trascurano le seguenti colonne:

- **ROW_ID**;
- **DOD_HOSP**;
- **DOD_SS**.

```
patient_filtered_df = patient_df.drop('ROW_ID', axis=1)
patient_filtered_df = patient_filtered_df.drop('DOD_HOSP',
    axis=1)
patient_filtered_df = patient_filtered_df.drop('DOD_SSN',
    axis=1)
```

Ottenendo un dataframe filtrato di **46520** righe e **5** colonne:

| SUBJECT_ID | GENDER | DOB | DOD | EXPIRE_FLAG |
|------------|--------|---------------------|---------------------|-------------|
| 249 | F | 2075-03-13 00:00:00 | NaN | 0 |
| 250 | F | 2164-12-27 00:00:00 | 2188-11-22 00:00:00 | 1 |
| 251 | M | 2090-03-15 00:00:00 | NaN | 0 |
| 252 | M | 2078-03-06 00:00:00 | NaN | 0 |
| 253 | F | 2089-11-26 00:00:00 | NaN | 0 |
| ... | ... | ... | ... | ... |

Tabella 2.3: Dataframe estratto da Patients

2.1.3 Dataframe finale estratto da NOTEVENTS e PATIENTS

Si procede all'unione dei due dataframe estratti nelle sezioni precedenti per ottenere il dataframe finale estratto da NOTEVENTS e PATIENTS su cui verranno effettuate tutte le operazioni descritte nel prosieguo del documento. In particolare, l'implementazione proposta per la creazione del dataframe finale è la seguente:

```
total_df = pd.merge(filtered_df, patient_filtered_df,
    on='SUBJECT_ID', how='inner')
```

Ottenendo un dataframe finale filtrato di **16120** righe e **8** colonne:

| ROW_ID | SUBJECT_ID | CHARTDATE | TEXT | GENDER | DOB | DOD | EXPIRE_FLAG |
|--------|------------|------------|-------------------------------------|--------|----------------|----------------|-------------|
| 175 | 13702 | 2118-06-14 | Admission Date: [**2118-6-2**] ... | F | 2037-05-03 ... | 2126-10-06 ... | 1 |
| 176 | 13702 | 2119-05-25 | Admission Date: [**2119-5-4**] ... | F | 2037-05-03 ... | 2126-10-06 ... | 1 |
| 177 | 13702 | 2124-08-18 | Admission Date: [**2124-7-21**] ... | F | 2037-05-03 ... | 2126-10-06 ... | 1 |
| 182 | 56174 | 2118-08-12 | Admission Date: [**2118-8-10**] ... | F | 2073-12-25 ... | NaN | 0 |
| 183 | 56174 | 2118-12-09 | Admission Date: [**2118-12-7**] ... | F | 2073-12-25 ... | NaN | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... |

Tabella 2.4: Dataframe finale estratto da Note Events e Patients

3 Memorizzazione dei dati

Contenuti

| | | |
|-------|--|----|
| 3.1 | Tecnologie usate | 15 |
| 3.1.1 | MongoDB | 15 |
| 3.1.2 | Neo4J | 17 |
| 3.2 | Architettura finale di memorizzazione Big Data | 18 |
| 3.2.1 | Analisi di scalabilità | 19 |

3.1 Tecnologie usate

Si è scelto di utilizzare la tecnologia di memorizzazione MongoDB per quanto riguarda il merge tra NOTEVENTS e PATIENTS poiché la struttura di dati processata nella Data Extraction è semi-strutturata, in particolare c'è la colonna TEXT che contiene, come già detto, tutte le informazioni relative alla cartella clinica del paziente i-esimo. MongoDB è proprio un database orientato ai documenti, il che significa che può memorizzare dati semi-strutturati senza la necessità di definire uno schema rigido.

Si è optato per un database di tipo **documentale** perché esso permette il retrieval anche di specifici campi all'interno di un documento, a differenza di un database chiave/valore che invece restituisce solamente documenti nella loro interezza.

Questo può semplificare il processo di caricamento dei dati e fornire flessibilità nel modo in cui vengono rappresentati, essendo le note cliniche presenti in NOTEVENTS contenute all'interno del campo "TEXT".

Nelle fasi successive, inoltre, in output dal processo di NER+EL+RE il dato sarà memorizzato su **Neo4J**. Questo perchè la struttura dei dati è a grafo giacchè le entità e le relazioni estratte tra di loro possono essere rappresentate molto più facilmente tra nodi e connessioni.

3.1.1 MongoDB

MongoDB è un database NoSQL orientato ai documenti che offre scalabilità, flessibilità ed è in grado di gestire enormi quantità di dati. Verranno di seguito descritte le principali proprietà di MongoDB:

- **Orientato ai documenti:** MongoDB immagazzina i dati come documenti, che sono strutture dati composte da coppie di chiavi e valori. I documenti sono simili agli oggetti JSON in termini di struttura e leggibilità, e sono raggruppati in collezioni, simili a tabelle in un database relazionale;
- **Schema flessibile:** A differenza dei database relazionali, i documenti in MongoDB possono avere campi diversi, permettendo una maggiore flessibilità nell'organizzazione dei dati;
- **Scalabilità orizzontale automatica:** MongoDB supporta la replica di dati e lo **sharding**, consentendo di distribuire i dati su più server per migliorare le prestazioni e la tolleranza ai guasti, il che lo rende altamente scalabile e affidabile;
- **Supporto per le transazioni:** A partire da MongoDB 4.0, è possibile eseguire transazioni multi-documento, simili a quelle in un database relazionale, che garantiscono proprietà ACID (Atomicity, Consistency, Isolation, Durability);
- **Driver in molteplici linguaggi:** MongoDB fornisce driver per una serie di linguaggi di programmazione, tra cui Java, Python, Node.js, C++, ed altri;

- **Alta disponibilità:** MongoDB offre alta disponibilità attraverso l'uso di replica sets, che sono gruppi di database che mantengono lo stesso set di dati. Se il database principale fallisce, un altro membro del replica set può prendere il suo posto;
- **Atomicità:** MongoDB supporta atomicità a livello di singolo documento, il che significa che le operazioni su un singolo documento sono atomiche: o avvengono completamente, o non avvengono affatto;
- **Aggregazioni:** MongoDB supporta varie operazioni di aggregazione, tra cui MapReduce. Quest'ultimo accetta una pipeline di operatori, dove le operazioni di filtraggio dovrebbero essere poste all'inizio per una maggiore efficienza.

3.1.1.1 MongoDB Atlas

Per l'utilizzo di MongoDB, si è proceduti con la configurazione di MongoDB Atlas. Esso è un servizio di gestione di database fornito da MongoDB che offre una piattaforma di gestione per le implementazioni di MongoDB, sia su piccola che su larga scala.

Proprietà che si aggiungono a quelle enunciate in precedenza per MongoDB sono:

- **Compatibilità:** MongoDB Atlas è completamente compatibile con i driver MongoDB e i client di MongoDB;
- **Database gestito:** Atlas gestisce automaticamente l'infrastruttura per il database MongoDB. Questo include il provisioning di server e la configurazione di MongoDB;
- **Sicurezza:** Atlas implementa un gran numero di funzioni di sicurezza, tra cui l'autenticazione, il controllo dell'accesso, la crittografia dei dati in riposo, la rete privata virtuale, l'auditing di sicurezza;
- **Monitoraggio e avvisi:** MongoDB Atlas fornisce strumenti di monitoraggio in tempo reale e avvisi configurabili;
- **Multi-Cloud:** Atlas offre la capacità di distribuire il tuo database su più provider di servizi cloud (AWS, Google Cloud, Azure), fornendo una maggiore flessibilità e ridondanza.

3.1.1.2 Workflow: Data Collection → Python → MongoDB Atlas

Come già anticipato, i dati da gestire si trovano all'interno dei file "NOTEVENTS.csv" e "PATIENTS.csv", presi dal database MIMIC-III, che contengono informazioni dettagliate su molteplici cartelle cliniche dei pazienti. I file hanno una dimensione di circa 4 GB e contengono più di 2 milioni di righe.

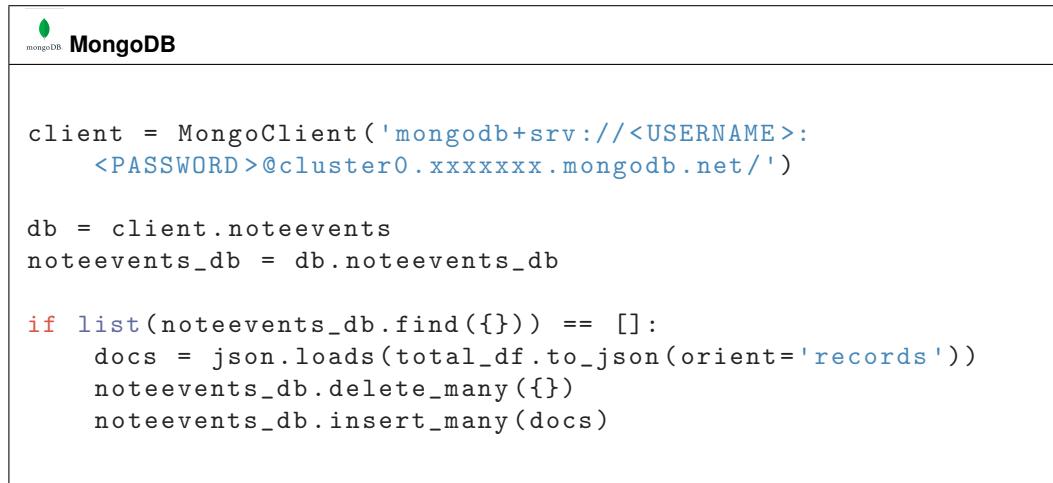
Avendo deciso di utilizzare MongoDB Atlas per l'archiviazione dei nostri dati è stato necessario effettuare le dovute operazioni di Extraction viste nel Capitolo 2, che hanno ridotto notevolmente la dimensione dei file, permettendo di lavorare con i dati in modo più efficiente. Infatti, **MongoDB Atlas ha un limite di dimensione del cluster di 512 MB**¹, questo limite ha influito nella scelta del numero di documenti da utilizzare per le fasi successive.

Una volta completate le operazioni preliminari, è stato utilizzato il driver MongoClient di MongoDB e la libreria PyMongo che, con uno script Python, ha permesso di connettersi al cluster di database Atlas, andando quindi a creare un **database** chiamato "noteevents" ed una **collezione documentale** all'interno di questo database chiamata "noteevents_db".

Prima di inserire i dati estratti nella collezione, si procede con la verifica dell'esistenza di essa e del suo contenuto; se non è vuota, la collezione non viene sovrascritta. Infine, viene convertito il dataframe ottenuto dalla fase preliminare in una lista di documenti JSON al fine di inserirli nella collezione "noteevents_db".

¹MongoDB Atlas mette a disposizione **512 MB gratuiti** per la memorizzazione di basi di dati usati solo per scopi didattici e illustrativi

Di seguito, il codice **Python** utilizzato:



```
client = MongoClient('mongodb+srv://<USERNAME>:<PASSWORD>@cluster0.xxxxxxxx.mongodb.net/')

db = client.noteevents
noteevents_db = db.noteevents_db

if list(noteevents_db.find({})) == []:
    docs = json.loads(total_df.to_json(orient='records'))
    noteevents_db.delete_many({})
    noteevents_db.insert_many(docs)
```

Tabella 3.1: Workflow: Data Collection - Python - MongoDB Atlas

Dopo aver caricato i dati, la collezione "noteevents_db" risultante ha una dimensione effettiva di archiviazione di 92.08 MB, con 16.120 documenti e una dimensione media del documento di 5.7 KB.

Inoltre, il database ha un solo indice, con una dimensione totale di 472 KB. Queste dimensioni ridotte rispetto ai file originali di 4 GB hanno permesso di utilizzare MongoDB Atlas senza superare il limite di dimensione del cluster di 512 MB.

3.1.2 Neo4J

Neo4j è un database a grafo nativo ad alte prestazioni per dati fortemente correlati. Esso si basa sulla teoria dei grafici ed utilizza una struttura di dati composta da nodi e relazioni.

I nodi rappresentano le entità nel database, mentre le relazioni, che sono direzionate e tipizzate, collegano i nodi tra loro, rappresentando come queste entità sono collegate o interagiscono tra loro. Entrambi, nodi e relazioni, possono avere proprietà, che sono coppie chiave-valore di informazioni aggiuntive.

Verranno di seguito descritte le principali proprietà di Neo4j:

- **Modello di grafico nativo:** Neo4j è una base dati a grafo nativo che consente di modellare i dati in modo intuitivo e flessibile senza simulazioni;
- **Linguaggio di interrogazione grafica (Cypher):** Neo4j utilizza un linguaggio di interrogazione specifico per i grafici chiamato Cypher, che rende facile l'interrogazione e la manipolazione dei dati nel grafo;
- **Alte prestazioni:** Neo4j è ottimizzato per velocità e scalabilità. È in grado di gestire interrogazioni complesse e computazionalmente onerose con elevata efficienza;
- **Transazionalità ACID:** Come i database relazionali, Neo4j supporta le transazioni ACID (Atomicità, Coerenza, Isolamento, Durabilità) per singole transazioni, garantendo l'affidabilità dei dati;
- **Schemaless:** Neo4j non presenta uno schema fisso, permettendo quindi di aggiungere al grafo nuovi nodi e proprietà.

3.2 Architettura finale di memorizzazione Big Data

In definitiva, si mostra l'architettura finale di memorizzazione del progetto che verrà esteso nei successivi paragrafi. È suddiviso tra **MongoDB** e **Neo4J**. In particolar modo i dati filtrati nella fase di *Data Extraction* sono memorizzati su MongoDB nel database noteevents nella collezione noteevents_db. Quest'ultimi verranno processati e trattati per la summarization e memorizzati nella collezione summaries_db ed estratti ambedue su Streamlit. Gli stessi dati processati verranno opportunamente trattati per la fase di NER+EL+RE che estraggono ulteriori colonne e relazioni che verranno memorizzate su Neo4J ed opportunamente trattate e visualizzate su Streamlit.

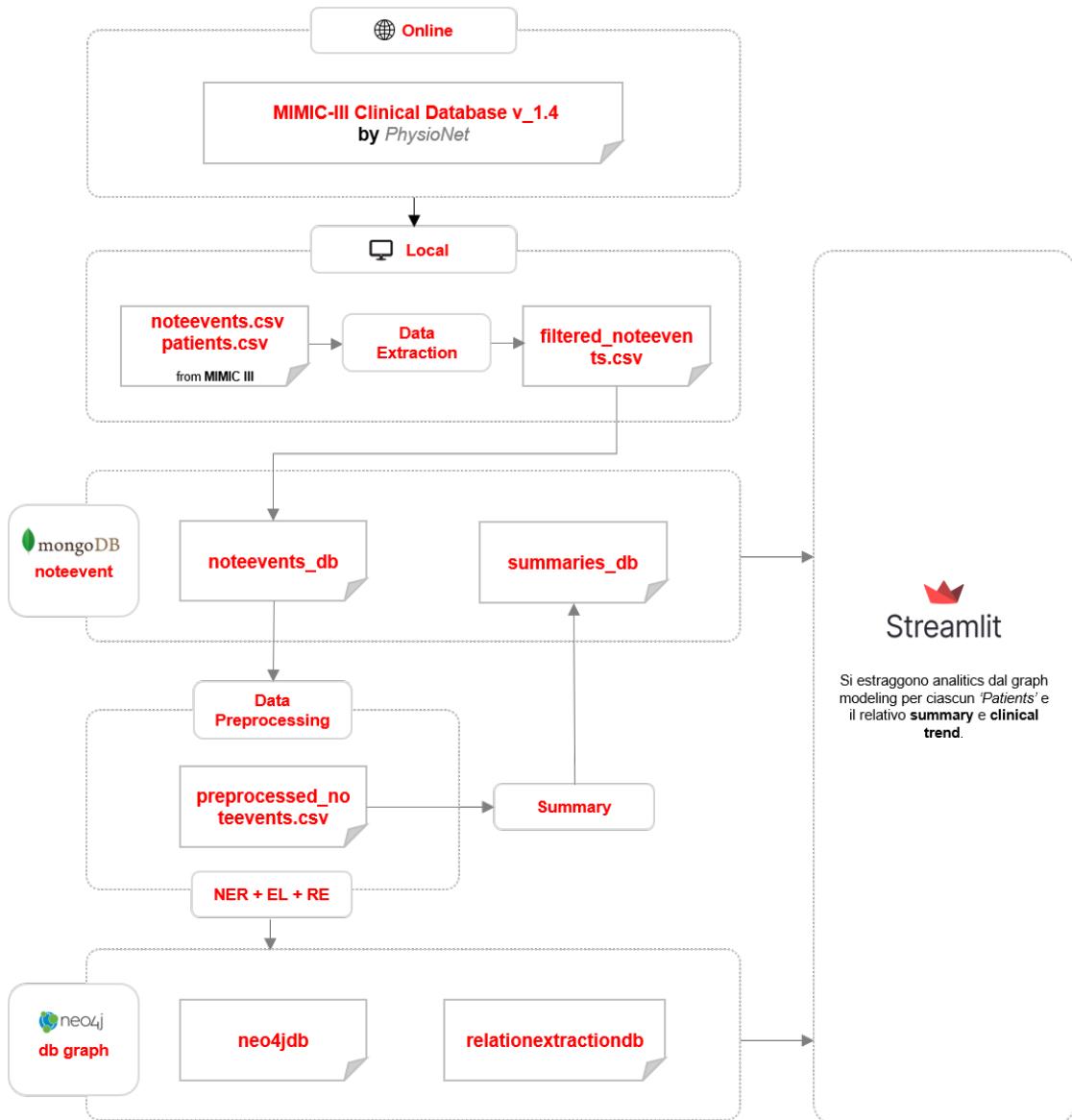


Figura 3.1: Architettura di Memorizzazione

3.2.1 Analisi di scalabilità

A partire dall'architettura di memorizzazione di sopra è possibile capire come avviene la gestione del carico di lavoro e la gestione dimensionale del progetto completo. Si elencano alcuni punti considerati nell'analisi di scalabilità:

- **Vincoli e limitazioni:** Non c'è stato nessuno investimento nell'architettura di memorizzazione, pertanto si è limitati alle offerte FREE delle tecnologie prescelte.
 - Dimensione massima di archiviazione su MongoDB: 512 MB e circa 16 MB massimo per ogni documento.
 - Supporto di una singola istanza di database su un singolo nodo macchina su Neo4J (versione Community Free)
- **Carico di lavoro:** Il sistema in esame prevede di elaborare il dataset MIMIC III le cui dimensioni sono circa 6-7 GB, ancora facilmente utilizzabile su una macchina locale. Il dataset, come trattato nei paragrafi precedenti, viene opportunamente filtrato e diminuito ai fini dei limiti imposti dal budget richiesto dall'architettura di memorizzazione cloud. Non è previsto un grande flusso di richieste al sistema.
- **Scalabilità:** Dall'architettura è possibile definire:
 - Scalabilità orizzontale: La piattaforma **Mongo DB** ha come feature principale proprio la scalabilità orizzontale che viene realizzata attraverso il concetto di sharding. Lo sharding consiste nella divisione dei dati in frammenti chiamati shard e nella distribuzione di questi shard su diversi server o cluster di server. Ogni shard contiene solo una porzione dei dati totali, consentendo un'elaborazione parallela e una migliore distribuzione del carico. **Neo4J**, nel caso in esame, non prevede scalabilità orizzontale.

4 Data preprocessing

Contenuti

| | |
|---|----|
| 4.1 Selezione dei pazienti | 20 |
| 4.1.1 Implementazione | 20 |
| 4.2 Section Extraction | 22 |
| 4.2.1 Implementazione | 22 |
| 4.3 Stopwords and Lemmatization | 23 |
| 4.3.1 Implementazione | 23 |
| 4.4 Pulizia delle note cliniche | 24 |
| 4.4.1 Implementazione | 24 |

Il preprocessing dei dati è una fase fondamentale nella preparazione di qualsiasi analisi o modello predittivo basato sui dati. Nel **contesto** dell'analisi dei testi clinici, il preprocessing dei dati comprende diverse operazioni che mirano a migliorare la qualità e la rappresentazione dei testi in modo da renderli adatti all'elaborazione. In questo capitolo, si visioneranno alcune delle tecniche di preprocessing dei dati comuni applicate nel caso in esame.

4.1 Selezione dei pazienti

Ai fini dell'esame e per l'esecuzione completa del progetto si fa la selezione dei pazienti, in accordo con quanto riportato nella traccia, vengono **scelti 20 pazienti (+ 1 paziente di esempio)**. Per effettuare tale selezione vengono applicati filtri e criteri specifici per identificare i pazienti che soddisfano determinate condizioni.

4.1.1 Implementazione

Nell'esempio fornito, vengono mostrate diverse iterazioni di selezione dei pazienti utilizzando il modulo PyMongo per interrogare la collezione documentale memorizzata su Mongo Atlas.

Viene creato un array vuoto chiamato **subject_ids** per memorizzare gli identificatori dei soggetti selezionati. Vengono quindi eseguite query separate sul database per selezionare i pazienti in base a intervalli specifici di SUBJECT_ID.

Ogni query utilizza il metodo \$match per specificare il range degli identificatori dei soggetti desiderati, seguito dal \$group per raggruppare i risultati in base all'_id del SUBJECT_ID. Infine, viene utilizzato il metodo \$limit per limitare il numero di risultati restituiti.

Per ogni query, i SUBJECT_ID distinti vengono estratti dai risultati e aggiunti all'array subject_ids. Infine, viene aggiunto manualmente un identificatore di soggetto specifico all'array come esempio. L'array subject_ids contiene gli identificatori dei pazienti selezionati.

Segue l'implementazione scritta in **Python**:

```
subject_ids = []
distinct_subjects = noteevents_db.aggregate([
    {'$match': {'SUBJECT_ID': {'$gte': 0, '$lte': 10000}}},
    {'$group': {'_id': '$SUBJECT_ID'}},
    {'$limit': 5}
])

# Aggiungi i SUBJECT_ID distinti all'array
for doc in distinct_subjects:
    subject_ids.append(doc['_id'])

distinct_subjects = noteevents_db.aggregate([
    {'$match': {'SUBJECT_ID': {'$gte': 10000, '$lte':
        20000}}},
    {'$group': {'_id': '$SUBJECT_ID'}},
    {'$limit': 5}
])

# Aggiungi i SUBJECT_ID distinti all'array
for doc in distinct_subjects:
    subject_ids.append(doc['_id'])

distinct_subjects = noteevents_db.aggregate([
    {'$match': {'SUBJECT_ID': {'$gte': 20000, '$lte':
        30000}}},
    {'$group': {'_id': '$SUBJECT_ID'}},
    {'$limit': 5}
])

# Aggiungi i SUBJECT_ID distinti all'array
for doc in distinct_subjects:
    subject_ids.append(doc['_id'])

distinct_subjects = noteevents_db.aggregate([
    {'$match': {'SUBJECT_ID': {'$gte': 30000, '$lte':
        40000}}},
    {'$group': {'_id': '$SUBJECT_ID'}},
    {'$limit': 5}
])

# Aggiungi i SUBJECT_ID distinti all'array
for doc in distinct_subjects:
    subject_ids.append(doc['_id'])
```

4.2 Section Extraction

L'estrazione delle sezioni è un'operazione che mira a estrarre porzioni rilevanti di testo all'interno di documenti clinici, infatti, le note cliniche presenti in MIMIC-III presentano molte sezioni spesso anche con informazioni ridondanti e/o superflue.

4.2.1 Implementazione

Nell'esempio fornito, viene utilizzato il modulo **MedSpaCy** [3] per eseguire l'estrazione delle sezioni. Tale modulo è una libreria di strumenti per l'esecuzione di attività di NLP cliniche e di elaborazione del testo con il popolare framework SpaCy. Il pacchetto MedSpaCy riunisce una serie di altri pacchetti, ognuno dei quali implementa funzionalità specifiche per l'elaborazione di testi clinici comuni, come la segmentazione delle frasi, l'analisi contestuale e l'asserzione degli attributi e il rilevamento delle sezioni.

Dopo aver caricato il modello MedSpaCy¹, viene aggiunto un componente chiamato "**medspacy_sectionizer**" alla pipeline del modello. Questo componente è responsabile dell'individuazione delle sezioni all'interno del testo clinico sulla base della sua conoscenza. La funzione **section_extraction** prende in input un testo e restituisce in output il testo suddiviso nelle sezioni estratte. Viene creato un oggetto doc utilizzando il modello **nlp()** e vengono iterati i titoli delle sezioni all'interno di **doc_.section_titles**. In base al titolo della sezione, viene aggiunto il testo corrispondente alla variabile **note_prep**. Al termine dell'estrazione delle sezioni dalla nota clinica, la funzione restituisce il testo delle sezioni estratte come un'unica nota preprocessata.

Segue l'implementazione scritta in **Python**:

```
def section_extraction(text):
    doc = nlp(text)
    i = 0
    note_prep = ""
    for title in doc_.section_titles:
        if str(title).lower() == 'chief complaint':
            note_prep += str(doc_.section_spans[i])
        if str(title).lower() == 'history of present
            illness:' or str(title).lower() == 'history':
            note_prep += str(doc_.section_spans[i])
        if str(title).lower() == 'past medical history':
            note_prep += str(doc_.section_spans[i])
        if str(title).lower() == 'discharge medications':
            note_prep += str(doc_.section_spans[i])
        if str(title).lower() == 'brief hospital course:
            or str(title).lower() == 'hospital course':
            note_prep += str(doc_.section_spans[i])
        if str(title).lower() == 'discharge diagnoses':
            note_prep += str(doc_.section_spans[i])
        i += 1

    return note_prep
```

¹Al repository GitHub ufficiale del progetto sono presenti vari esempi che spiegano come usare il modulo MedSpaCy: <https://github.com/medspacy/medspacy>

4.3 Stopwords and Lemmatization

In questa sezione verranno discusse tecniche di rimozione delle stopwords ed esecuzione della lemmatizzazione. Le stopwords sono parole comuni, come articoli, preposizioni e congiunzioni, che di solito non portano informazioni significative per l'analisi dei testi. La rimozione di queste parole può contribuire a ridurre la dimensione del corpus di testo e a focalizzare l'attenzione sulle parole più rilevanti. La lemmatizzazione, d'altra parte, è il processo di riduzione delle parole alle loro forme di base o lemmi. Questo passaggio semplifica l'analisi dei testi, consentendo di trattare tutte le varianti di una parola come una singola entità.

4.3.1 Implementazione

Per eseguire questa operazione, viene utilizzato il modulo NLTK², il quale mette a disposizione diverse librerie come quella contenente l'elenco delle stopwords in lingua inglese, una libreria che effettua la tokenizzazione di un testo e la libreria basata su WordNet che effettua la lemmatizzazione.

In particolare, la fase di preprocessing segue la seguente pipeline:

- **Tokenizzazione:** la nota clinica viene divisa in **token**, ovvero in parole singole;
- **Rimozione delle stopwords:** vengono rimosse le **stopwords** (articoli, avverbi, congiunzioni, ecc.) e i simboli indesiderati;
- **Lemmatizzazione:** le parole vengono riportate alla loro forma base (**lemma**), ad esempio verbi coniugati vengono riportati all'infinito, ecc.

Segue l'implementazione completa scritta in **Python**:

```
def tokenize_and_lemmatize(text):  
    # Tokenizzazione delle parole  
    tokens = word_tokenize(text)  
    # Inizializzazione del lemmatizer  
    lemmatizer = WordNetLemmatizer()  
  
    # Rimozione delle stopwords e simboli  
    stop_words = set(stopwords.words('english'))  
    symbols = ['[', '*', '+']  
    filtered_tokens = [token for token in tokens if  
        token.lower() not in stop_words and ... and not  
        any(symbol in token for symbol in symbols)]  
  
    # Lemmatizzazione delle parole  
    lemmatized_tokens = [lemmatizer.lemmatize(token) for  
        token in filtered_tokens]  
    # Riaccorpamento del testo lemmatizzato  
    lemmatized_text = ' '.join(lemmatized_tokens)  
  
    return lemmatized_text
```

²Natural Language Toolkit: <https://www.nltk.org/>

4.4 Pulizia delle note cliniche

Pertanto il processo di stopwords, lemmatization e section extraction definisce il preprocessing e la pulizia delle note cliniche.

4.4.1 Implementazione

Dunque vanno semplicemente chiamate le due funzioni definite precedentemente **section_extraction** e **tokenize_and_lemmatize** per tutte le note cliniche dei pazienti selezionati.

Al fine di effettuare la pulizia come detto sopra viene, quindi, iterato il dataframe df per applicare la pulizia a ogni nota. Innanzitutto, vengono estratte dalle note le sezioni d'interesse, dopodiché la nota estratta viene salvata in una nuova colonna del dataframe chiamata "**CLEANED TEXT**" (**Codice 1**).

Segue l'implementazione scritta in **Python**:

```
# Creazione di una nuova colonna per il testo pulito
df['CLEANED TEXT'] = ''

# Iterazione sulle righe del DataFrame
for index, row in df.iterrows():
    # Ottenimento del testo dalla colonna desiderata (es.
    # 'Text')
    text = row['TEXT']

    # Estrazione delle sezioni dal testo
    cleaned_text = section_extraction(text)

    # Salvataggio del testo pulito nella nuova colonna
    # 'Cleaned Text'
    df.at[index, 'CLEANED TEXT'] = cleaned_text
```

Tabella 4.1: Codice 1: Section extraction

Nello step successivo il testo delle note, con le sezioni estratte, viene lemmatizzato e salvato nella nuova colonna del dataframe **"LEMMATIZED TEXT"** (**Codice 2**).

Segue l'implementazione scritta in **Python**:

```
# Creazione di una nuova colonna per il testo lemmatizzato
df['LEMMATIZED TEXT'] = ''

# Iterazione sulle righe del DataFrame
for index, row in df.iterrows():
    # Ottenimento del testo dalla colonna desiderata
    text = row['CLEANED TEXT']

    # Tokenizzazione e lemmatizzazione del testo
    lemmatized_text = tokenize_and_lemmatize(text)

    # Salvataggio del testo lemmatizzato nella nuova
    # colonna 'Lemmatized Text'
    df.at[index, 'LEMMATIZED TEXT'] = lemmatized_text
```

Tabella 4.2: Codice 2: Lemmatization

I passaggi effettuati per il preprocessing delle note sono fondamentali per rendere i dati pronti per l'analisi e l'estrazione di informazioni significative nelle fasi successive.

5 Generative Language Model

Contenuti

| | | |
|-------|--|----|
| 5.1 | Prompt | 27 |
| 5.2 | Summary | 27 |
| 5.2.1 | Il Modello | 28 |
| 5.3 | Clinical Trend | 29 |
| 5.3.1 | Il Modello | 30 |
| 5.4 | Salvataggio dei Summary e dei Clinical Trend | 31 |

Un **Generative Language Model** è un tipo di modello di intelligenza artificiale, addestrato per generare testo in linguaggio naturale simile a quello umano. Questo tipo di modello è in grado di apprendere le caratteristiche, la struttura e il contesto del testo attraverso l'addestramento su un vasto corpus di dati. Nel contesto di questo progetto, il **GLM** viene utilizzato per generare un riassunto della storia di un paziente utilizzando le note cliniche associate a quel paziente. Al modello viene fornito in input il **prompt**: un testo strutturato con indicazioni ed esempi che permette al modello di produrre in output un riassunto che descrive in modo accurato, ma conciso, la storia medica del paziente, le condizioni attuali e i dettagli rilevanti, a partire dalle condizioni attuali ed andando a ritroso nel tempo. Per la scelta del miglior modello GLM di OpenAI si è valutato tra i seguenti:

| Modelli OpenAI | Tempi di esecuzione (s) | Risultato | Costi |
|------------------|-------------------------|---|----------------------|
| davinci | 423 | Non riesce a scrivere il testo completamente, ripetitivo | \$0.0200 / 1K tokens |
| text-davinci-002 | 354 | A volte non è preciso e non completa il riassunto, si dilunga | \$0.0200 / 1K tokens |
| text-davinci-003 | 386 | Si dilunga e va oltre i limiti, ma non è conciso | \$0.0200 / 1K tokens |
| GPT 3.5 turbo | 298 | Conciso e riassume completamente nei limiti imposti | \$0.002 / 1K tokens |

Tabella 5.1: Scelta dei modelli Open AI

La scelta pertanto è ricaduta su **GPT 3.5 Turbo**.

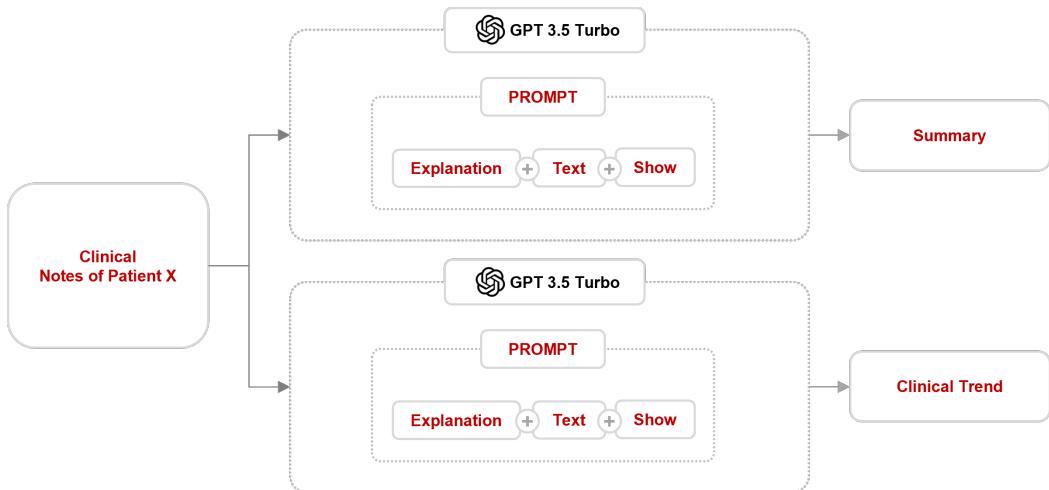


Figura 5.1: Architettura Clinical Notes a Summary e Clinical Trend

5.1 Prompt

Il prompt fornito al GLM include istruzioni specifiche su come generare un riassunto a partire dalle note cliniche dei pazienti, in particolare un prompt può essere espresso come **la somma** delle seguenti componenti:

$$P = E + T + S \quad (5.1)$$

dove:

- **P = Prompt** finale;
- **E = Explanation:** spiegazione di ciò che si vuole dal modello;
- **T = Text:** testo che il modello deve trattare e riassumere;
- **S = Show:** dimostrazione di un esempio di output utile al modello;

Verranno forniti due prompt differenti, il primo servirà a generare il **Summary** [8] della storia clinica del paziente, mentre, il secondo fornirà il **Clinical Trend**, ovvero un termine riassuntivo sulle condizioni cliniche del paziente caratterizzata dalla sua evoluzione temporale. Con entrambi i prompt, verrà richiesto al modello di concentrarsi su un singolo paziente alla volta e sulla sua storia medica, evitando ripetizioni inutili e dettagli irrilevanti come prescrizioni e dosaggi di farmaci. Il prompt include anche indicazioni sul formato desiderato del riassunto, come ad esempio il numero massimo di parole usate per generare il testo.

5.2 Summary

Un summary di note cliniche può essere utile per fornire una panoramica rapida e concisa delle informazioni rilevanti contenute nelle note cliniche di un paziente. Questo riassunto condensa i dati essenziali sulla storia medica, le diagnosi, i trattamenti e i risultati dei test, consentendo ai professionisti sanitari di ottenere rapidamente una visione complessiva dello stato di salute del paziente. Il summary delle note cliniche facilita la comunicazione tra i membri del team medico, risparmia tempo nella valutazione dei dati e aiuta a prendere decisioni informate sulle cure e sulle strategie terapeutiche da adottare.

5.2.0.1 Prompt

Nel prompt relativo alla Summarization, pertanto, sono state utilizzate le seguenti componenti:

- **Explanation:**
 - Generate a complete but concise (max 100 words) and informative summary that focuses only on the unique patient, that is always the same throughout the notes in input, and his medical history, current condition, and relevant details, starting from his current conditions backwards;
 - The summary must be accurate and avoid unnecessary repetition, avoid details of prescriptions, doses and other subjects besides the specific patient;
 - If there are dates, enter the year, month and the day; Do not include doctors' names;
 - It must be easy for a doctor to understand, the tone is formal; The summary always starts with "The patient is a:".
- **Text**

- NOTE 1 etc.
- NOTE 2 etc.
- ...
- NOTE N etc.

● **Show** the type of the task:

- The patient currently is a age-year-old gender who presented with chief complaint and has a medical history of relevant medical conditions.
- The patient was involved in incident/accident description, which led to specific injuries/traumas;
- The postoperative course involved relevant procedures performed on anatomical locations involved;
- The patient is currently prescribed medications for specific purposes;

5.2.1 Il Modello

Al modello GPT 3.5 Turbo, oltre a sottomettergli il prompt, ha necessariamente bisogno anche del tipo di ruolo ¹ che deve intraprendere nella generazione del sommario. In particolare nel caso in esame si ha il ruolo del:

- System: "*You are a formal medical assistant specialising in the summary of a patient's clinical notes*"
- User: "*prompt*"

Si riporta l'estratto dell'implementazione per la summarization del testo in **Python**:

```
for subject_id, lemmatized_texts in grouped_data.items():
    prompt_notes = ""
    for i, text in enumerate(lemmatized_texts):
        prompt_notes += f"\n Note {i+1}: {text}"

    prompt = f""" {prompt_notes} + Explanation + Show the
                  type of the task
"""

# MODELLO DI GPT PER LA SUMMARIZATION
message = [{"role": "system", "content": "You are a
              formal medical assistant specialising in the summary
              of a patient's clinical notes."},
            {"role": "user", "content": prompt}]
completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=message,
    max_tokens=256,
    stop=None,
    temperature=0.3,
    frequency_penalty=0.7
)
```

¹Oltre al ruolo del **System** e dell'**User** c'è il ruolo dell'**Assistant** che è proprio la risposta che il generatore dovrà costruire

La parte di codice relativa alla Summarization riceve in input un file CSV (preprocessed_noteevents.csv) contenente i dati delle note cliniche preprocessate, come viste nel Capitolo precedente, di vari pazienti.

Per ogni paziente presente nel dataframe vengono raggruppate le sue note cliniche e vengono aggiunte nel prompt.

Il prompt viene inviato al modello tramite la chiamata all'API **openai.ChatCompletion.create()**. Il modello genererà una risposta che rappresenta il riassunto richiesto. Un esempio è il seguente:

SUBJECT_ID: 27431

SUMMARY: The patient is a 77-year-old male nursing home resident with a medical history of schizophrenia, CAD, HTN, and dementia. He presented with hypoxia and hypotension, which led to intubation and CVL placement. He had a recent admission for similar symptoms and was found to have osteomyelitis. The patient also had a history of aspiration pneumonia and chronic decubitus ulcers. Despite treatment with antibiotics, he remained persistently hypoxic and suffered PEA arrest leading to death.

5.3 Clinical Trend

Il Clinical Trend valuta la direzione del cambiamento nel quadro clinico del paziente sulla base delle informazioni riportate nella nota. Esso fornisce un'indicazione se le condizioni del paziente sono migliorate (**improvement**), peggiorate(**worsening**), rimaste stabili(**stable**) o morto(**dead**) nel periodo di tempo considerato.

5.3.0.1 Prompt

Per quanto riguarda l'estrazione del Clinical Trend, il prompt fornito prevede le seguenti componenti:

● **Explanation:**

- Identify in 1 word the clinical trend like this: extract a single word that accurately represents the clinical trend observed in the patient's notes, considering every detail and keyword;
- For the word, use the general indicators: 'Improvement', 'Stable', 'Worsening' to describe the trend;
- Otherwise identify in 1 word: "Dead" if the patient from his notes is dead.

● **Text**

- NOTE 1 etc.
- NOTE 2 etc.
- . . .
- NOTE N etc.

● **Show** the type of the task:

- "Improvement".

5.3.1 Il Modello

Anche in questo caso al modello GPT 3.5 Turbo, oltre a sottomettergli il prompt, c'è bisogno di indicargli il ruolo che deve intraprendere nella generazione del sommario. In particolare si ha il ruolo del:

- System: " You are a clinical trend extractor of a patient 's clinical notes . "
- User: "prompt"

Si riporta l'estratto dell'implementazione per la summarization del testo in **Python**:

```
for subject_id, lemmatized_texts in grouped_data.items():
    prompt_notes = ""
    for i, text in enumerate(lemmatized_texts):
        prompt_notes += f"\n Note {i+1}: {text}"
    prompt = f""" {prompt_notes} + Explanation + Show the
                    type of the task
"""

# MODELLO DI GPT PER LA SUMMARIZATION
message = [{"role": "system", "content": "You are a
    clinical trend extractor of a patient's clinical
    notes."},
            {"role": "user", "content": prompt}]
completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=message,
    max_tokens=5,
    stop=None,
    temperature=0.3,
    frequency_penalty=0.7
)
```

La parte di codice relativa all'estrazione del Clinical Trend per ogni paziente nel dataframe raggruppa le sue note e crea il relativo prompt che include tutte le note associate a quel paziente, insieme alle istruzioni per l'estrazione della tendenza clinica. Viene utilizzato il modello GPT-3.5 Turbo di OpenAI per generare il Clinical Trend, attraverso una conversazione simulata tra un estrattore di tendenze cliniche e il modello. Il prompt viene inviato al modello tramite la chiamata all'API **openai.ChatCompletion.create()**. Il modello genera una risposta che rappresenta il trend clinico estratto.

SUBJECT_ID: 27431
CLINICAL TREND: Dead

5.4 Salvataggio dei Summary e dei Clinical Trend

Infine, la successiva implementazione prevede la combinazione dei risultati dei **summary** e dei **clinical trend** in un unico dataframe e il salvataggio del dataframe nella collezione documentale "**summaries_db**" in MongoDB:

```
merged_df = pd.merge(summary_df, clinical_df,
                     on="SUBJECT_ID")
client = MongoClient('mongodb+srv://  
    USERNAME:PASSWORD@healthcare.maxcbgn.mongodb.net/')
db = client.noteevents
summaries_db = db.summaries_db
for subject in summary_df['SUBJECT_ID']:
    if summaries_db.find_one({'SUBJECT_ID': subject}) is
        None:
        doc =
            json.loads(merged_df.to_json(orient='records'))
        summaries_db.insert_many(doc)
```

Questo dataframe contiene le informazioni dell'ID del paziente, il summary e il clinical trend estratto per ciascun paziente. Successivamente, viene stabilita una connessione con il database MongoDB Atlas, visto in precedenza, utilizzando il modulo PyMongo. Viene selezionato il database "noteevents" e la collezione documentale "summaries_db".

6 Named Entity Recognition

Contenuti

| | | |
|-------|--|----|
| 6.1 | MedCAT | 32 |
| 6.1.1 | Implementazione | 33 |
| 6.2 | Relation Extraction | 41 |
| 6.2.1 | Prompt - Relation Extraction | 41 |
| 6.2.2 | Implementazione | 42 |

La **Named-Entity Recognition** (NER) è un processo chiave nel campo dell'elaborazione del linguaggio naturale, che consiste nell'individuazione e l'etichettatura di entità significative all'interno di un testo. Nel contesto dei testi clinici, le entità possono essere nomi di malattie, procedure mediche, farmaci, nomi dei pazienti e altre informazioni rilevanti per la diagnosi, il trattamento o la documentazione medica. L'obiettivo della **NER** è quello di estrarre queste entità in modo automatico e accurato, consentendo una migliore comprensione dei testi clinici e fornendo input per ulteriori analisi o applicazioni. La NER può essere realizzata utilizzando approcci basati su regole, tecniche di apprendimento automatico o modelli di linguaggio preaddestrati. Nel caso in esame si è scelto di utilizzare il modello **MedCAT** in combinazione con **MedMentions** perché tale modello è risultato quello con le prestazioni migliori per il task richiesto, le metriche di valutazione tenute in conto sono le seguenti ¹:

| Model / Dataset | MedMentions Expanded | | | MedMentions Strict | | | MedMentions Strict (Diseases) | | |
|----------------------|----------------------|--------|----------|--------------------|--------|----------|-------------------------------|--------|----------|
| | Precision | Recall | F1 Score | Precision | Recall | F1 Score | Precision | Recall | F1 Score |
| MedCAT MM | 0.613 | 0.807 | 0.710 | - | - | - | - | - | - |
| MedCAT U / MI | 0.414 | 0.427 | 0.420 | 0.425 | 0.694 | 0.559 | 0.675 | 0.907 | 0.791 |
| MedCAT U / MM | 0.434 | 0.525 | 0.479 | 0.422 | 0.858 | 0.640 | 0.720 | 0.977 | 0.848 |

Tabella 6.1: Metriche di valutazione MedCAT

6.1 MedCAT

Medical Concept Annotation Toolkit (MedCAT) [5] è un modulo utile per estrarre informazioni dalle cartelle cliniche elettroniche (EHR) e collegarle a ontologie biomediche ² come SNOMED-CT e UMLS.

MedCAT viene utilizzato per la Named-Entity Recognition (**NER**), focalizzandosi nel dominio medico, dunque nel contesto dei testi clinici. È stato progettato specificamente per l'elaborazione del linguaggio naturale nel campo della medicina e offre funzionalità avanzate per l'estrazione automatizzata e l'etichettatura di entità cliniche rilevanti. Si basa su tecniche di apprendimento automatico e modelli di linguaggio preaddestrati (come **SpaCy**) per l'**individuazione e l'etichettatura delle entità all'interno dei testi clinici**, infatti, grazie a MedCAT è stato possibile effettuare anche l'**Entity Linking**.

MedCAT utilizza un **database di concetti** (Concept Database, **CDB**) che contiene informazioni sulle diverse categorie di entità cliniche, come malattie, procedure mediche, farmaci e altri concetti correlati alla diagnosi e al trattamento medico. Il CDB può essere personalizzato e adattato alle specifiche esigenze del dominio medico in cui viene utilizzato.

¹I valori di Precision, Recall e F1 Score sono stati presi dal paper di MedCAT: <https://arxiv.org/pdf/1912.10166.pdf>

²Un'ontologia biomedica è un insieme di concetti, relazioni e regole che rappresentano il dominio di conoscenza biomedicale.

La libreria **MedCAT** offre anche la possibilità di integrare modelli di linguaggio preaddestrati, come BERT o altri modelli di deep learning, per migliorare ulteriormente le prestazioni del riconoscimento delle entità, tuttavia in questo progetto tali approcci non sono stati usati.

L'architettura utilizzata per effettuare la **NER** e l'**Entity Linking** è la seguente:

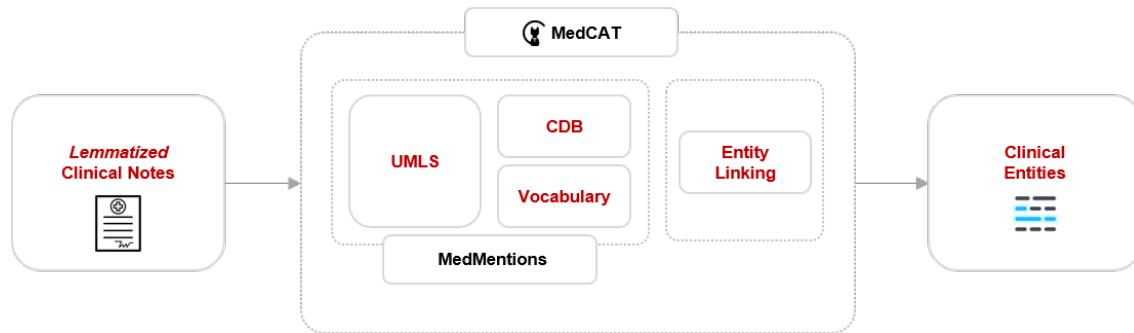


Figura 6.1: Architettura NER + EL

6.1.1 Implementazione

Si procede con la descrizione dettagliata dell'implementazione ³:

1. Importazione delle librerie necessarie per l'esecuzione del codice: Più nel dettaglio:

```

from medcat.cat import CAT
from medcat.config import Config
from medcat.utils.vocab import Vocab
from medcat.cdb import CDB
from spacy import displacy
  
```

- **CAT (Concept Annotation Tool):** È la classe principale di MedCAT utilizzata per l'annotazione dei concetti. CAT utilizza un modello di apprendimento automatico preaddestrato per riconoscere le entità nel testo medico. Può essere configurato con un database di concetti (CDB), un vocabolario e categorie aggiuntive (MetaCAT) per migliorare il processo di annotazione;
- **CDB (Concept Database):** È una classe che rappresenta un database di concetti. Un CDB è un componente fondamentale di MedCAT e contiene informazioni sulle entità, come nomi, descrizioni, sinonimi e relazioni semantiche. Il CDB viene utilizzato da CAT per il riconoscimento delle entità e per fornire informazioni aggiuntive sui concetti;
- **Vocab (Vocabulary):** È una classe che rappresenta un vocabolario di parole utilizzato da MedCAT. Il vocabolario viene utilizzato per mappare le parole nel testo a concetti specifici nel CDB. Aiuta a migliorare la precisione del riconoscimento delle entità e a gestire le ambiguità nel testo medico;

³Per informazioni più dettagliate sull'utilizzo di MedCAT si faccia riferimento al repository GitHub ufficiale del progetto: <https://github.com/CogStack/MedCAT>

- **MetaCAT (Meta Categorical Annotation Tool):** È una classe che fornisce informazioni aggiuntive sulle categorie delle entità. Ad esempio, può essere utilizzata per etichettare le entità con lo stato, come **"Affirmed"** o **"Other"**. Il MetaCAT viene utilizzato per migliorare il processo di annotazione delle entità e per filtrare le entità in base alle categorie specificate. Nel caso in esame è molto importante discriminare le entità "Affirmed" o "Other" confermando o meno la presenza di quell'entità nella rispettiva cartella clinica;
 - **SpaCy:** È una libreria open-source per il linguaggio naturale che fornisce funzionalità per il preprocessamento dei testi, il riconoscimento delle entità nominate (NER) e altre operazioni di analisi del linguaggio. Nel codice, viene utilizzata per la visualizzazione delle entità riconosciute tramite il modulo displacy.
2. Viene definita la cartella principale dei dati DATA_DIR e vengono creati i percorsi dei file di vocabolario vocab_path e del database di concetti cdb_path utilizzando la cartella dei dati:

```
DATA_DIR = "./data/"  
vocab_path = DATA_DIR + "vocab.dat"  
cdb_path = DATA_DIR + "cdb-medmen.dat"
```

Si noti: Per il caso in esame si utilizza come base di conoscenza **MedMentions** [6], un corpus di documenti biomedici annotati con menzioni di entità appartenenti a UMLS (Unified Medical Language System). Questo corpus contiene circa 35.000 concetti medici ed il suo modello MetaCAT è stato **costruito a partire proprio da un campione del MIMIC-III**.

3. Creazione e caricamento del CDB (Concept Database), del Vocabolario e del modello per le MetaAnnotations:

```
cdb = CDB.load(cdb_path)  
vocab = Vocab.load(vocab_path)  
mc_status = MetaCAT.load("./data/Status/")
```

Vengono dunque creati un oggetto cdb utilizzando il metodo load() della classe CDB per caricare il database di concetti, un oggetto vocab utilizzando il metodo load() della classe Vocab per caricare il vocabolario, ed il modello per le MetaAnnotations utilizzando il percorso specificato.

4. Creazione di CAT, la classe principale di MedCAT:

```
cat = CAT(cdb=cdb, config=cdb.config,  
vocab=vocab, meta_cats=[mc_status])
```

Viene creato un oggetto cat utilizzando la classe CAT di MedCAT. Vengono passati il database di concetti cdb, la configurazione di cdb, il vocabolario vocab e le MetaCategories.

5. Definizione dei tipi di entità da riconoscere ⁴:

```
type_ids_filter = ['T023', 'T034', 'T046',
                    'T047', 'T048', 'T059', 'T060', 'T061',
                    'T121', 'T184']
```

Vengono definiti gli ID dei tipi di entità che si desidera riconoscere. Questi ID corrispondono alle diverse categorie di entità all'interno del database di concetti:

- bpoc - T023 - **Body Part, Organ, or Organ Component**;
- lbtr - T034 - **Laboratory or Test Result**;
- patf - T046 - **Pathologic Function**;
- dsyn - T047 - **Disease or Syndrome**;
- mobd - T048 - **Mental or Behavioral Dysfunction**;
- lbpr - T059 - **Laboratory Procedure**;
- diap - T060 - **Diagnostic Procedure**;
- topp - T061 - **Therapeutic or Preventive Procedure**;
- phsu - T121 - **Pharmacologic Substance**;
- sosy - T184 - **Sign or Symptom**.

6. Filtraggio dei **CUI** (Concept Unique Identifier) basato sui tipi di entità:

```
cui_filters = set()
for type_ids in type_ids_filter:
    cui_filters.update(cat.cdb.addl_info
                        ['type_id2cuis'][type_ids])
cat.cdb.config.linked['filters']['cuis'] =
    cui_filters
```

Viene creato un insieme **cui_filters** e viene iterato su ogni ID di tipo di entità nel filtro. Per ogni ID, vengono estratti i CUI corrispondenti dal database di concetti e vengono aggiunti all'insieme.

Infine, viene aggiornato il filtro dei CUI nella configurazione del database di concetti **cat.cdb.config.linked** **['filters']****['cuis']** con l'insieme dei CUI filtrati.

⁴Tutte le entità riconosciute dal modello sono presenti al seguente link: https://lhncbc.nlm.nih.gov/ii/tools/MetaMap/Docs/SemanticTypes_2018AB.txt

7. Definizione di un dizionario dei colori per le categorie di entità:

```
color_dict = {
    'Body Part, Organ, or Organ Component':
        '#FFCE80',
    'Laboratory or Test Result': '#FFF9C4',
    'Disease or Syndrome': '#B5EAD7',
    'Mental or Behavioral Dysfunction': '#F0B2FF',
    'Laboratory Procedure': '#DOD9FF',
    'Diagnostic Procedure': '#FFD9EC',
    'Therapeutic or Preventive Procedure':
        '#C4FFFF',
    'Pharmacologic Substance': '#FFDAB9',
    'Sign or Symptom': '#FFC4F3',
    'Pathologic Function': '#B5EAD7'
}
```

Viene definito un dizionario **color_dict** che associa ogni categoria di entità a un colore specifico. Questo verrà utilizzato successivamente per la visualizzazione delle entità riconosciute.

8. Esecuzione del riconoscimento delle entità per il testo di esempio:

```
entities_lemma = cat.get_entities(data['LEMMATIZED
TEXT'][0])
```

Viene utilizzato il metodo **get_entities** dell'oggetto cat per riconoscere le entità nel testo lemmatizzato specificato. Il risultato viene salvato nella variabile entities_lemma.

9. Estrazione dei concetti riconosciuti:

```

extracted_data = []
for key, value in
    entities_lemma['entities'].items():
        if value['meta_anno']['Status']['value'] ==
            'Affirmed':
            if value['types'][0] == 'Pharmacologic
                Substance':
                extracted_data.append((
                    value['source_value'],
                    value['types'][0]))
            else:
                extracted_data.append((
                    value['pretty_name'],
                    value['types'][0]))

```

Viene creato un elenco di concetti estratti dalle entità riconosciute nel testo lemmatizzato. Solo le entità con lo stato **"Affirmed"** vengono considerate. Se l'entità è di tipo "Pharmacologic Substance", viene utilizzato il valore **"source_value"** come nome, altrimenti, viene utilizzato il **"pretty_name"**.

I concetti estratti vengono poi inseriti in un dataframe df che presenta le colonne "Nome" e "Entità", come mostrato di seguito:

| Nome | Entità |
|---------------------------|----------------------------------|
| Hypoxia | Pathologic Function |
| Schizophrenia | Mental or Behavioral Dysfunction |
| Coronary Arteriosclerosis | Disease or Syndrome |
| Hypertensive disease | Disease or Syndrome |
| Dementia | Mental or Behavioral Dysfunction |
| ... | ... |

Tabella 6.2: Annotazione su testo preprocessato

10. Visualizzazione delle entità riconosciute utilizzando **displacy**:

```

doc = {"text": data['LEMMATIZED_TEXT'][0], "ents": []
}, "title": None}

for key, value in
entities_lemma['entities'].items():
if value['meta_anns']['Status']['value'] ==
'Affirmed':
ent = {
'start': value['start'],
'end': value['end'],
'label': value['types'][0]
}
doc['ents'].append(ent)

colors = color_dict
options = {"ents": list(colors.keys()), "colors":
colors}

displacy.render(doc, style='ent', options=options,
jupyter=True, manual=True)

```

Viene creato un oggetto doc con il testo lemmatizzato e un elenco vuoto di entità riconosciute. Viene poi popolato l'elenco delle entità utilizzando i risultati del riconoscimento delle entità precedente. Viene definito il dizionario dei colori colors e le opzioni per la visualizzazione delle entità. Infine, viene chiamato il metodo render di displacy per visualizzare le entità riconosciute.



Figura 6.2: Recognized Entities

11. Viene inizializzata una lista risultati_finali per salvare i risultati finali delle entità riconosciute in ogni riga del dataframe 'data'. Viene iterato su ogni riga del dataframe e viene utilizzato il metodo `get_entities` di cat per riconoscere le entità nel testo lemmatizzato della riga corrente. I risultati vengono quindi elaborati e aggiunti alla lista risultati_finali. Infine, i risultati vengono concatenati in un unico dataframe df_completo e salvati in un file CSV chiamato "ner_noteevents.csv":

```
risultati_finali = []
# Itera su ogni riga del dataframe 'data'
for index, row in data.iterrows():
    entities_tot = cat.get_entities(row['LEMMATIZED TEXT'])

    extracted_data = []
    for key, value in entities_tot['entities'].items():
        if value['meta_anns']['Status']['value'] == 'Affirmed':
            if value['type_ids'][0] in type_ids_filter:
                if value['types'][0] == 'Pharmacologic Substance':
                    extracted_data.append((
                        value['source_value'],
                        value['types'][0]))
        else:
            extracted_data.append((
                value['pretty_name'],
                value['types'][0]))

    . . .

    df_finale['Subject ID'] = row['SUBJECT_ID']
    df_finale['Note ID'] = row['ROW_ID']
    risultati_finali.append(df_finale)

df_completo = pd.concat(risultati_finali)
df_completo.to_csv("ner_noteevents.csv", sep=";", index=False)
```

Si schematizzano gli step della **NER + EL**, mostrati sopra, attraverso il seguente **workflow** completo di **MedCAT**:

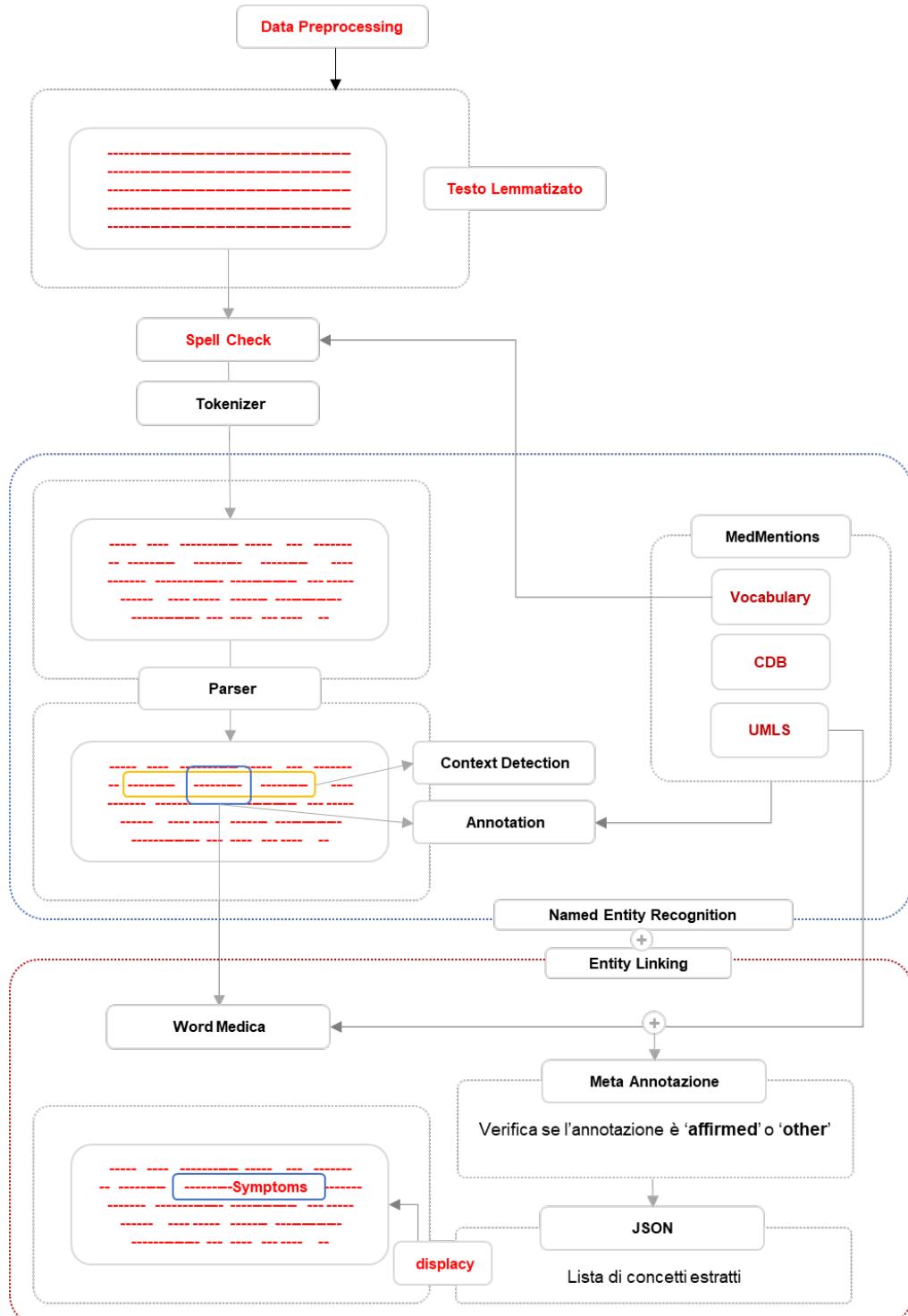


Figura 6.3: Workflow NER + EL

6.2 Relation Extraction

A partire dalle entità e concetti estratti nella fase di **NER+EL**, si è deciso di trovare delle relazioni tra essi. La relazione tra i concetti medici può essere importante per la diagnosi e il trattamento di malattie. Ad esempio, la relazione tra un sintomo e una condizione medica può aiutare a identificare la causa del sintomo e il trattamento appropriato. Inoltre, la relazione tra una procedura diagnostica e un risultato di laboratorio può aiutare a interpretare il risultato del test e a formulare una diagnosi accurata.

Si è utilizzato il modello **GPT 3.5 Turbo** per estrarre le relazioni a partire dalle entità estratte nella fase di **NER+EL**. Il workflow seguito per l'estrazione delle relazioni è il seguente:



Figura 6.4: Architettura RE

6.2.1 Prompt - Relation Extraction

Anche nel caso dell'estrazione delle relazioni, si usa dunque la Prompt Engine per "indirizzare" il modello GPT 3.5 a svolgere questo determinato compito.

Il prompt utilizzato per automatizzare la Relation Extraction è composto dalle seguenti componenti principali:

● **Explanation:**

- Find the relation between Disease or Sindrome, Diagnostic Procedure, Sign or Symptom, Pharmacologic Substance and Laboratory or Test Result;
- Generate at most 1 relation for each disease in the list, do not generate concepts that are not present in the list and leave the cell blank when the concept are not present;
- The first row always be: "Disease or Syndrome, Diagnostic Procedure, Sign or Symptom, Laboratory or Test Result, Pharmacologic Substance".

● **Text:**

- Disease or Sindrome = [...]
- Diagnostic Procedure = [...]
- Sign or Symptom = [...]
- Pharmacologic Substance = [...]
- Laboratory or Test Result = [...]

- **Show** the type of the task:

– The output must be formatted as csv: Transient Ischemic Attack, Electrocardiography, Chest Pain, Partial pressure CO₂ result, atorvastatin.

6.2.2 Implementazione

A partire dal file CSV "ner_noteevents.csv" (si ricorda che esso contiene dati relativi ai concetti medici estratti, le cui colonne fanno riferimento al tipo/categoria di concetto medico, ad esempio **Body Part**, **Organ**, or **Organ Component**, **Laboratory or Test Result**, **Diagnostic Procedure**, **Disease or Syndrome**, **Laboratory Procedure**, **Mental or Behavioral Dysfunction**, **Pathologic Function**, **Pharmacologic Substance**, **Sign or Symptom**, **Therapeutic or Preventive Procedure**), si vuole mostrare come verranno estratte le relazioni tra i concetti medici.

I passaggi seguiti sono:

- eliminazione di colonne del dataframe "df" che non sono rilevanti per l'analisi e creazione di un dizionario chiamato "medical_concepts" in cui verranno conservati i concetti medici per ogni riga;
- si itera sulle righe del dataframe e, per ciascuna riga, viene creato un dizionario vuoto chiamato "row_concepts" per conservare i concetti medici per ogni categoria;
- si itera sulle colonne della riga e si controlla se il concetto esiste e non è 'NaN'. Se il concetto esiste, lo si aggiunge al dizionario "row_concepts" per la categoria corrispondente;
- infine, si itera sui concetti medici delle righe e si crea un prompt di testo per ogni riga che contiene i concetti medici estratti. Il prompt di testo viene utilizzato per interrogare un modello di linguaggio di intelligenza artificiale fornito da OpenAI per trovare la relazione tra i concetti medici nella riga.

Segue l'implementazione scritta in **Python**:

```
df = pd.read_csv('ner_noteevents.csv', sep=';', header=0)
. .
df = df.drop('Body Part, Organ, or Organ Component', axis=1)
df = df.drop('Laboratory Procedure', axis=1)
. .
medical_concepts = {}
for index, row in df.iterrows():
    row_concepts = {}
    for column in df.columns:
        concept = row[column]
    . .
    medical_concepts[index] = row_concepts

for index, row_concepts in medical_concepts.items():
    prompt_text = ""
    for category, concepts in row_concepts.items():
        prompt_text += f"{category}: "
        prompt_text += ", ".join(concepts)
        prompt_text += "\n"
    prompt_text += """ + Explanation + Show the type of the
task"""

message = [{"role": "system", "content": "You are an
expert medical assistant."},
            {"role": "user", "content": prompt_text}]

completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=message,
    max_tokens=600,
    stop=None,
    temperature=0.3,
    frequency_penalty=0.7
)

completion_text =
    completion['choices'][0]['message']['content']
```

A seguito della **Relation Extraction** effettuata con il modello generativo GPT-3.5-turbo, occorre unire le relazioni estratte per ogni paziente in un unico file CSV in modo da automatizzare la successiva fase di caricamento e creazione del **Knowledge Graph**.

Segue l'implementazione di questa operazione di unione, scritta in **Python**:

```
result = "Subject ID,Note ID,Disease or Syndrome,Diagnostic  
Procedure,Sign or Symptom,Laboratory or Test  
Result,Pharmacologic Substance"  
result += '\n'  
  
for idx, row in id.iterrows():  
    with open(f'Relation Extracted/{idx+1}.txt', 'r') as f:  
        text = f.read()  
  
    lines = text.split('\n')  
    lines = lines[1:]  
  
    new_lines = [f'{id.loc[idx, "Subject ID"]},{id.loc[idx,  
"Note ID"]},{line}' for i, line in enumerate(lines)]  
  
    result += '\n'.join(new_lines)  
    result += '\n'  
  
with open('Relation Extracted/result.txt', 'w') as f:  
    f.write(result)  
  
data = pd.read_csv("Relation Extracted/result.txt",  
sep=',', on_bad_lines='skip')  
data.to_csv('Relation Extracted/result.csv', sep=',',  
index=False)
```

Quindi si riassumono gli step seguiti in questo Capitolo per la **NER+EL** e la **RE** nella seguente architettura completa:

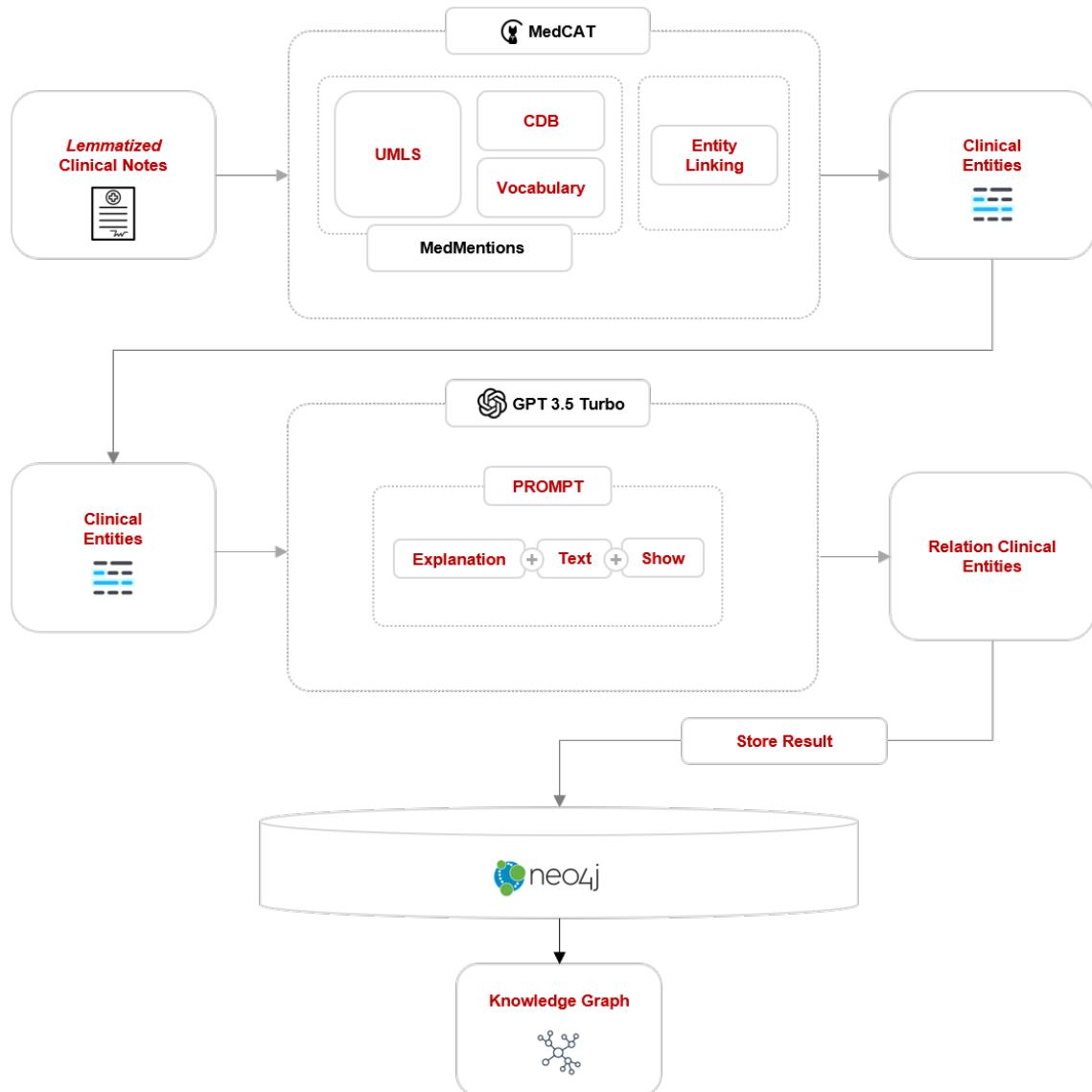


Figura 6.5: Architettura NER+EL e RE

7 Graph Modeling

Contenuti

| | | |
|-------|--|----|
| 7.1 | Memorizzazione su Neo4J | 46 |
| 7.2 | Concept Graph | 47 |
| 7.2.1 | Schema | 47 |
| 7.2.2 | Network esteso | 48 |
| 7.3 | Knowledge Graph | 49 |
| 7.3.1 | Schema | 49 |
| 7.3.2 | Network esteso | 49 |
| 7.3.3 | Entità collegate alla nota clinica | 50 |
| 7.3.4 | Relazioni per entità cliniche | 52 |
| 7.4 | Risultati | 53 |

L'obiettivo del Graph Modeling in esame è quello di rappresentare i dati clinici, con l'intento di creare un grafo che rappresenti le connessioni tra le diverse entità cliniche, consentendo una visualizzazione immediata delle relazioni per poter svolgere in maniera chiara e precisa delle analisi su di esse.

7.1 Memorizzazione su Neo4J

A valle del processo di **NER+EL+RE** si avrà il dataset con le entità estratte dalla NER+EL e il dataset con le relazioni estratte dal processo di RE. Quest'ultimi due saranno memorizzati sulla piattaforma **Neo4J** [4] per procedere alle rispettive operazioni di costruzione del Graph Modeling. In particolare si procede nell'implementazione di due grafi:

1. Il **Concept Graph** in cui si mostrano tutte le entità collegate direttamente alla Nota Clinica e il paziente;
2. Il **Knowledge Graph** in cui vengono creati nodi per le malattie ed altri tipi di entità, ma vengono anche stabiliti collegamenti specifici tra le entità.

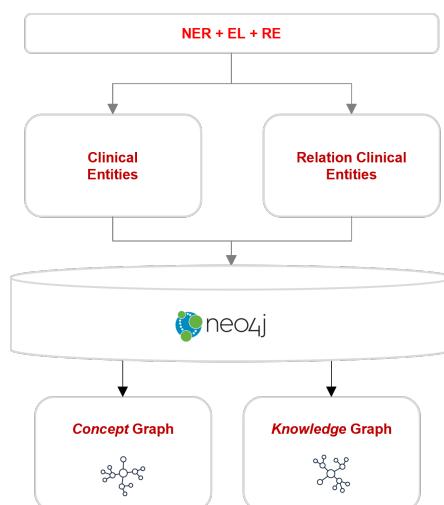


Figura 7.1: Dalla NER+EL+RE a Neo4J

7.2 Concept Graph

In un Concept Graph i nodi rappresentano i concetti estratti nella fase di **NER+EL** identificando le entità rilevanti come i *Sign or Symptoms*, *Disease or Syndrome*, *Diagnostic Procedure* etc... connesse all'i-esima nota del paziente.



Con la medesima interconnessione si riesce a capire nell'immediato, ad esempio, tutti i sintomi che sono stati rilevati nella nota clinica del paziente. Il limite però sta nella capacità della rete di capire quali possono essere le possibili relazioni che vi sono tra *Sign or Symptom*, *Disease or Syndrome* etc...

7.2.1 Schema

Si definisce il seguente schema dove è possibile apprezzare immediatamente l'interconnessione predominante tra paziente-note cliniche ed entità estratte.



Figura 7.2: Concept Graph Schema

7.2.2 Network esteso

Si estende lo schema nel network del Concept Graph completo. Non è immediata la visione del grafo ma è da qui che si riesce ad estrarre tutte le informazioni concettuali del paziente e nella rispettiva nota.

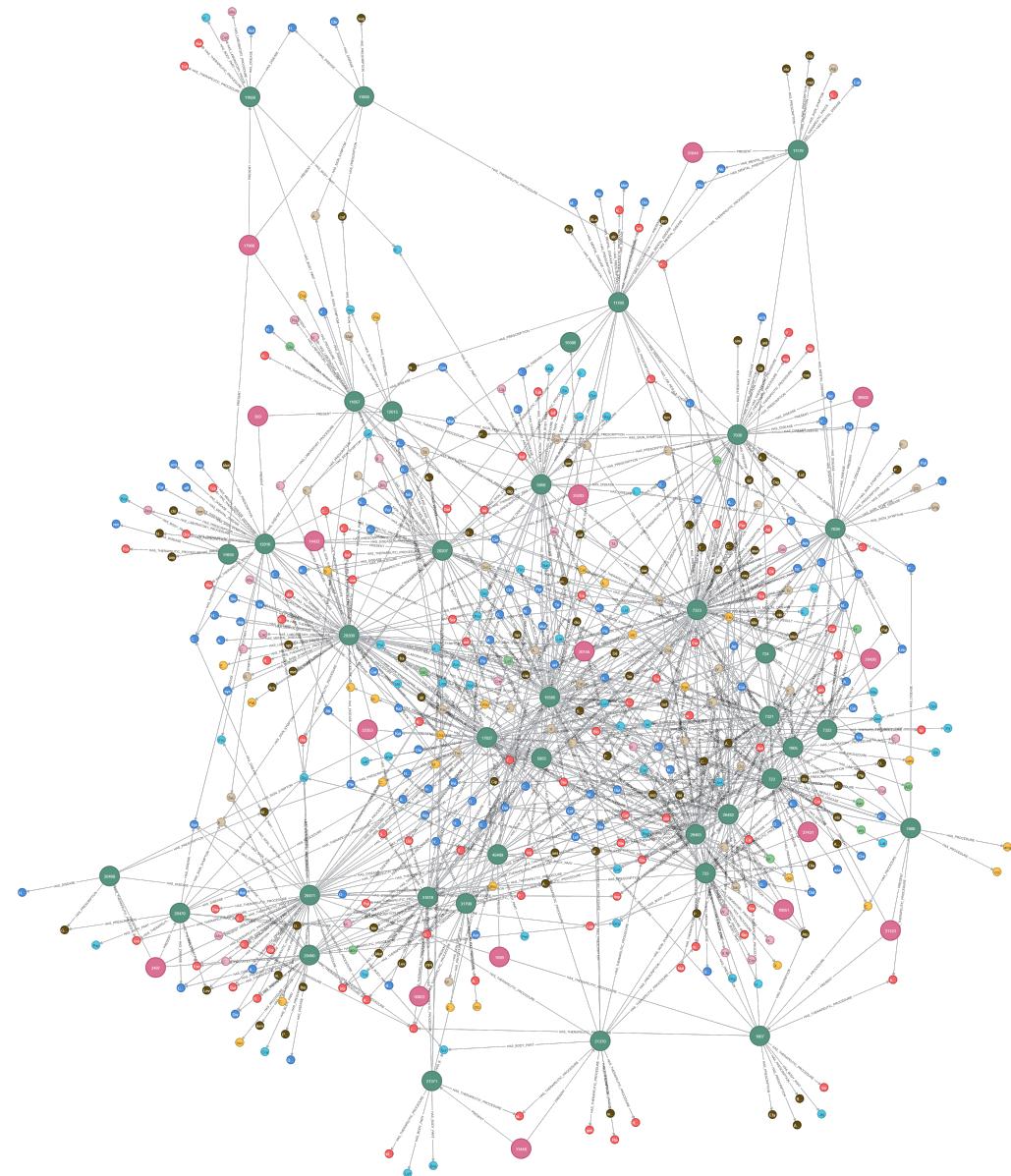
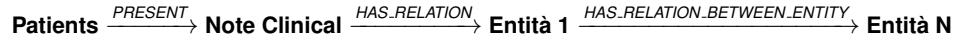


Figura 7.3: Concept Graph Network

7.3 Knowledge Graph

In un Knowledge Graph si estendono le relazioni anche tra le entità rilevate dalla **NER+EL**. Infatti, a valle del processo di **RE** (Relation Extraction) si ha un grafo complesso, strutturato e con tante sovra interconnessioni che rappresentano una conoscenza maggiore rispetto al grafo precedente.



7.3.1 Schema

Si definisce il seguente schema dove è possibile apprezzare la differenza con il Concept Graph.

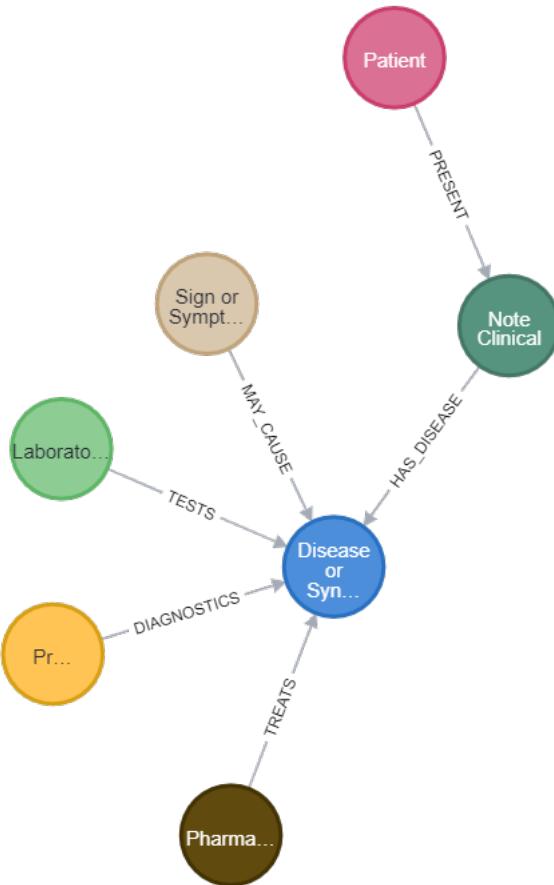


Figura 7.4: Knowledge Graph Schema

Il grafo mostrato raffigura soltanto la relazione nota clinica → malattia. Ma la differenza con il precedente grafo sta proprio nelle relazioni con le altre entità *Sign or Symptom*, *Procedure*, etc.

7.3.2 Network esteso

Si estende lo schema nel **network** del **Knowledge Graph**. In questo caso, è ancora più difficile apprezzare la visione del grafo dato le interconnessioni anche tra le entità estratte. La differenza immediata con il precedente network è sicuramente la maggior conoscenza della rete.

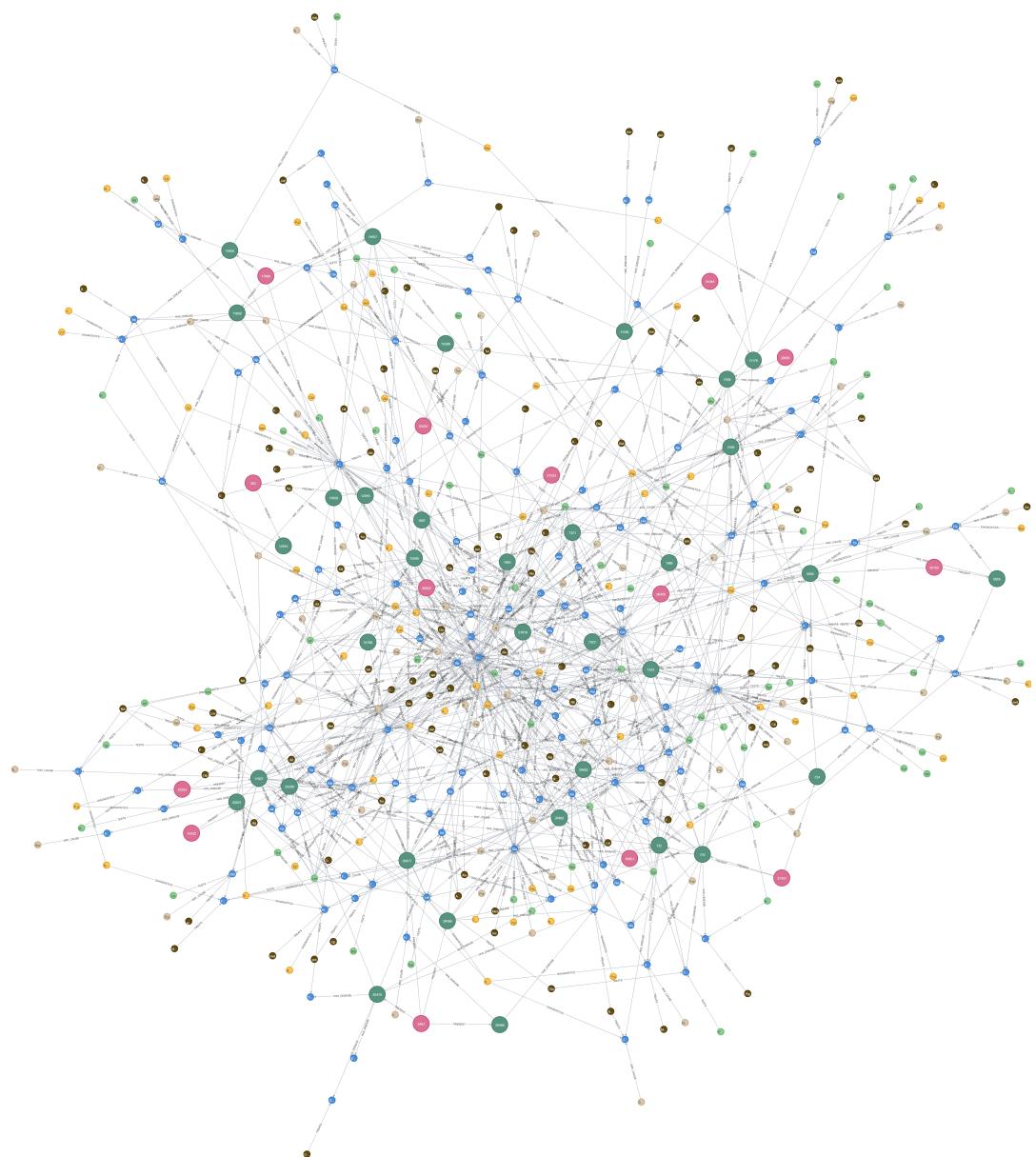


Figura 7.5: Knowledge Graph Network

A valle delle costruzioni dei grafi verranno fatte query i cui risultati saranno visualizzati nella dashboard interattiva completa.

7.3.3 Entità collegate alla nota clinica

In dettaglio, si effettuano le seguenti operazioni:

- Viene stabilita la connessione al database Neo4j, successivamente viene letto il file CSV 'ner_noteevents.csv', che ricordiamo contenere i concetti medici e la categoria a cui fanno parte;
- Vengono letti i valori delle colonne corrispondenti alle diverse entità cliniche, come procedure diagnostiche, parti del corpo, malattie, ecc., e assegnati a variabili corrispondenti;

- Si procede alla creazione dei nodi di tipo "Patient" nel grafo, si controlla innanzitutto se il nodo non esiste, poi si aggiunge un nuovo nodo di tipo "Patient" con l'ID del soggetto al grafo;
- Viene creato un nodo di tipo "Note Clinical" con l'ID della nota clinica (note_clinical_id) e si aggiunge un arco di relazione tra il nodo del paziente (subject_node) e il nodo della nota clinica (note_node) con l'etichetta "PRESENT";
- Si itera il processo di creazione dei nodi per tutte le entità estratte dalla NER.

Segue un estratto dell'implementazione della prima fase di Graph Modeling in **Python**:

```
graph = Graph("bolt://localhost:7687", user="neo4j",
              password="")
note = pd.read_csv('ner_noteevents.csv', sep=';', header=0)

for index, row in note.iterrows():
    subject_id = row['Subject ID']
    note_clinical_id = row['Note ID']
    ...
    drugs = row['Pharmacologic Substance']

    subject_node = graph.nodes.match("Patient",
                                      subject_id).first()
    if not subject_node:
        subject_node = Node("Patient", subject_id)
        graph.create(subject_node)

    note_node = Node("Note Clinical",
                     note_clinical_id=note_clinical_id)
    relationship = Relationship(subject_node, "PRESENT",
                                 note_node)
    graph.create(relationship)

    if isinstance(diagnostics, str):
        diagnostics = diagnostics.split(", ")
        for diagnostic in diagnostics:
            diagnostic_node = Node("Diagnostic Procedure",
                                   name=diagnostic)
            graph.merge(diagnostic_node, "Diagnostic
Procedure", "name")
            relationship = Relationship(note_node,
                                         "HAS_PROCEDURE", diagnostic_node)
            graph.create(diagnostic_node)
            graph.create(relationship)
    ...
    .
```

7.3.4 Relazioni per entità cliniche

Segue un approccio simile a quello precedente, con alcune differenti operazioni:

- Viene stabilita una connessione ad un database differente di Neo4j e viene letto il file CSV 'result.csv', che ricordiamo contenere le relazioni tra concetti medici estratte nel Capitolo precedente;
- Viene gestita la creazione dei nodi e delle relazioni per le malattie, i segni/sintomi, i risultati dei test di laboratorio e i farmaci utilizzando il file CSV 'result.csv';
- Viene creato un nodo di tipo "Disease" con il nome della malattia. Se il nodo esiste già, viene effettuato un merge. In base ai valori delle altre variabili, vengono creati nodi corrispondenti per i segni/sintomi, i risultati dei test di laboratorio e i farmaci, creando le opportune relazioni tra i suddetti nodi.

Segue un estratto dell'implementazione della seconda fase di Graph Modeling in **Python**:

```
graph_re = Graph("bolt://localhost:7687",
    name="relationextraction", user="neo4j", password="")
relation = pd.read_csv('result.csv', sep=',', header=0)

for index, row in relation.iterrows():
    subject_id = row['Subject ID']
    note_clinical_id = row['Note ID']
    .
    .
    drugs = row['Pharmacologic Substance']

    subject_node = graph_re.nodes.match("Patient",
        subject_id).first()
    if not subject_node:
        subject_node = Node("Patient", subject_id)
        graph_re.create(subject_node)

    note_node = Node("Note Clinical", note_clinical_id)
    graph_re.merge(note_node, "Note Clinical",
        "note_clinical_id")
    .

    if isinstance(diseases, str):
        diseases = diseases.split(", ")
        for disease in diseases:
            disease_node = Node("Disease or Syndrome",
                name=disease)
            .

            if isinstance(sign_symptoms, str):
                sign_symptoms_split =
                    sign_symptoms.split(", ")
                for sign_symptom in sign_symptoms_split:
                    .


```

7.4 Risultati

Attraverso l'implementazione precedente, è possibile osservare come vengono automaticamente creati i nodi e le relazioni su Neo4j in base ai dati presenti nei file CSV forniti.

Se si volessero osservare dei risultati grafici attraverso l'utilizzo di Neo4j:

- per la prima fase (trovare entità cliniche collegate alla nota clinica), è possibile trovare tutti i percorsi (path) che corrispondono a una relazione di tipo "**HAS_DISEASE**" tra due nodi, restituendo i percorsi trovati nella query come risultato;
 - per la seconda fase (trovare le relazioni tra le entità cliniche), è possibile eseguire la stessa query trovata nella prima fase, con la differenza che troverà una relazione tra nodi a partire dalle informazioni estratte nella Relation Extraction.

Segue il comando utilizzato per mostrare i risultati grafici sottostanti:

```
neo4j
MATCH p=()-[r:HAS_DISEASE]->() RETURN p LIMIT 25
```

Tabella 7.1: Graph Modeling - Cypher – Neo4j

Si noti come, nelle Figure sottostanti, per il grafo della prima fase (**Concept Graph**) per una specifica cartella clinica (nell'esempio la 722) sono associati tutti i concetti estratti dalla NER+EL, ovvero "Disease or Syndrome" (rappresentati in blu), "Laboratory Procedure" (rappresentati in rosa), "Diagnostic Procedure" (rappresentati in arancione), etc.

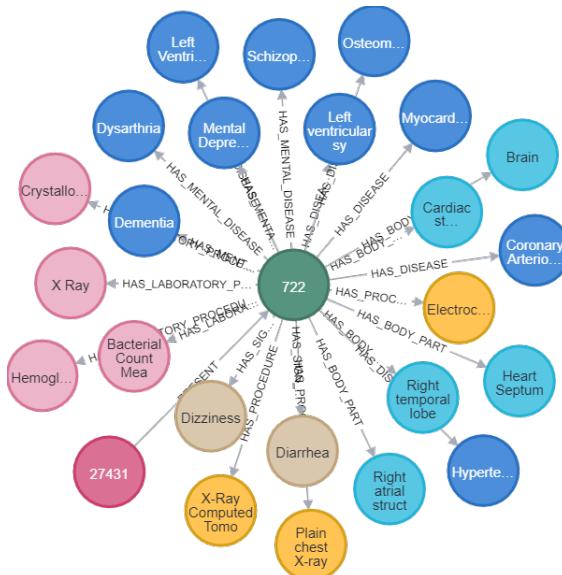


Figura 7.6: Entità cliniche collegate alla nota clinica

Mentre per il grafo della seconda fase (**Knowledge Graph**) si nota come per la stessa nota clinica usata per la Figura precedente (la 722) ad essa sono collegate solo i concetti di "Disease or Syndrome" (rappresentati in blu), mentre tutti gli altri concetti sono collegati mediante opportune relazioni a quest'ultimi, ad esempio abbiamo i nodi di tipo "Sign or Symptom" (rappresentati in beige) i quali sono relazionati ai "Disease or Syndrome" tramite un arco "MAY_CAUSE".

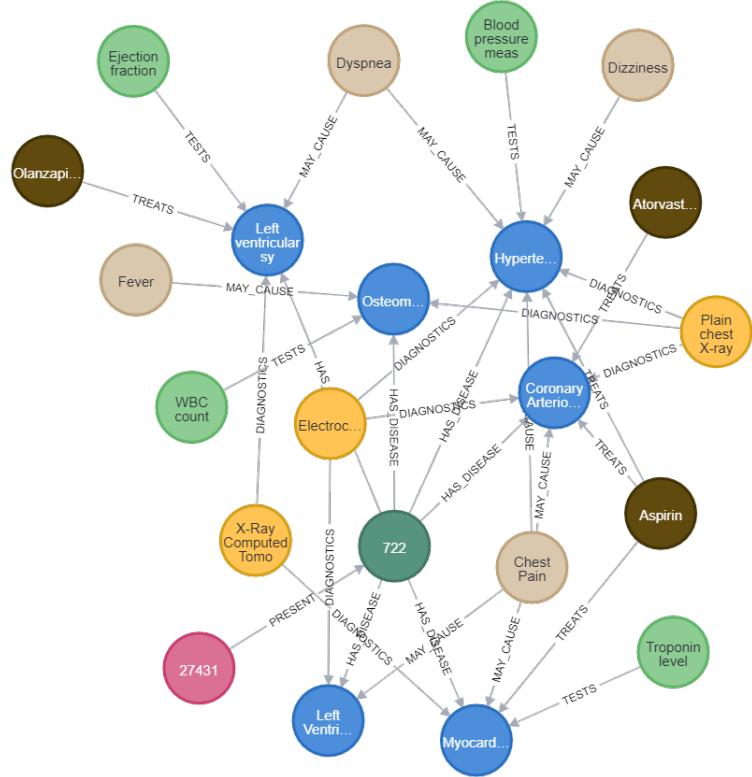


Figura 7.7: Relazione tra entità cliniche

8 Analytics and Report

Contenuti

| | |
|---|----|
| 8.1 Dashboard | 55 |
| 8.2 Analytics | 57 |
| 8.2.1 Informazioni demografiche e cliniche del paziente | 57 |
| 8.2.2 Estrazione Clinical Trend | 59 |
| 8.2.3 Estrazione Summary | 60 |
| 8.2.4 Numero di malattie/sintomi diagnosticati nel tempo | 61 |
| 8.2.5 Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo | 63 |
| 8.2.6 Elenco di farmaci, sintomi e diagnostiche | 65 |
| 8.2.7 Patologie del paziente associate a note cliniche | 67 |
| 8.2.8 Parti del corpo doloranti del paziente associate alle note cliniche | 68 |
| 8.2.9 Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche | 69 |
| 8.2.10 Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche | 70 |
| 8.2.11 Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche | 71 |
| 8.2.12 Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche | 72 |

In questo capitolo, l'obiettivo prefissato è quello di mostrare, tramite un interfaccia web, tutte le informazioni raccolte nei capitoli precedenti, utilizzando delle Analytics specifiche per mostrare risultati dettagliati per ciascun paziente, note cliniche, malattie ed altro. A supporto di questo bisogno si è utilizzato Streamlit, framework open-source in Python che semplifica la creazione di applicazioni web per l'analisi e la visualizzazione dei dati. Inoltre, con l'ausilio di alcune librerie, verranno estratti i dati caricati nelle operazioni precedenti dai database MongoDB e Neo4j per poi visualizzare le informazioni rilevanti su tale interfaccia web.

8.1 Dashboard

La fase di creazione della dashboard comprende diverse operazioni di configurazione e connessione a diverse fonti di dati. Vediamo i principali punti di questa fase:

- **Configurazione della pagina:** Vengono specificati il titolo della pagina, l'icona da visualizzare e il layout desiderato;
- **Configurazione per AGraph:** Viene creata una configurazione personalizzata per la visualizzazione di un grafo utilizzando AGraph;
- **Connessione a MongoDB:** Viene definita una funzione che si connette a un database MongoDB utilizzando la libreria PyMongo e che consente di prelevare i dati dalla collezione summaries_db;
- **Connessione a Neo4j:** Viene definita una funzione che si connette a un database Neo4j utilizzando la libreria py2neo. Vengono ottenuti due oggetti Graph per interagire con il database e restituiti come risultato della funzione;

- **Sidebar:** Viene configurata la sidebar. Viene aggiunto un'intestazione e viene creato un selettore per selezionare il paziente;
- **Caricamento dei dati dai filtered_noteevents:** Viene definita una funzione che si connette nuovamente al database MongoDB per ottenere i dati dei filtered_noteevents relativi al paziente selezionato;
- **Caption:** Viene aggiunta una caption che fornisce informazioni aggiuntive sui nomi, data di nascita, (eventuale) data di morte, data di ultima ammissione in un ospedale e numero di note cliniche associate ai pazienti nel dataset MIMIC-III.

La pagina principale della dashboard avrà il seguente aspetto:

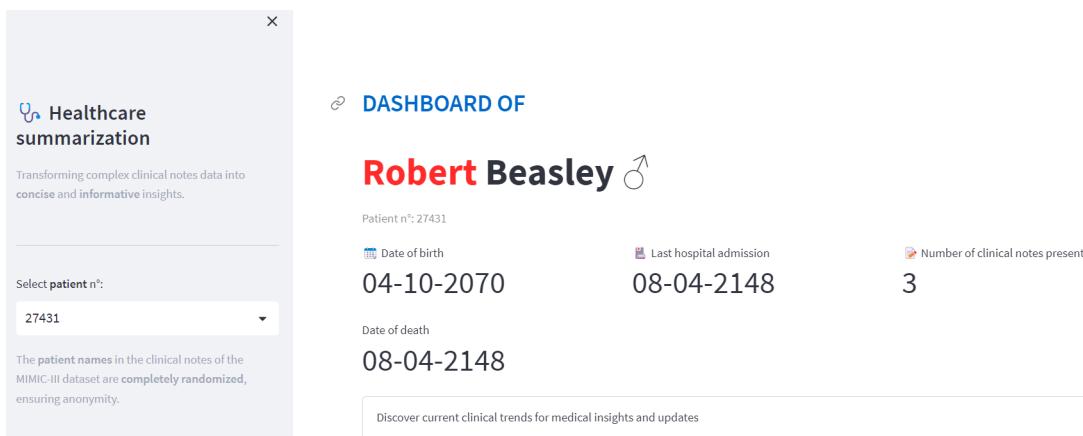


Figura 8.1: Dashboard

Si noti: Per ottenere le informazioni sui pazienti, si riporta alla prima delle Analytics presenti nella sezione successiva.

8.2 Analytics

Di seguito verranno mostrate diverse Analytics sui dati a disposizione relativi a note cliniche, patologie, diagnosi, sintomi ed altro di ciascun paziente selezionato nella sidebar precedentemente discussa. Da ciò, è stato possibile ottenere un quadro dettagliato dei pazienti. In questa raccolta di Analytics, vengono selezionate le analisi più significative che permetteranno di comprendere al meglio il quadro clinico dei pazienti.

8.2.1 Informazioni demografiche e cliniche del paziente

L'obiettivo della prima Analytic è quello di mostrare un'analisi sul paziente selezionato, fornendo informazioni demografiche e cliniche rilevanti. In questo modo, viene fornita una panoramica delle informazioni essenziali sul paziente, come la data di nascita, l'ultimo ricovero ospedaliero, il numero di note cliniche presenti e, se applicabile, la data di morte.

8.2.1.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
dob = noteevents_df['DOB'].head(1).to_string(index=False)
dob = datetime.strptime(dob, "%Y-%m-%d %H:%M:%S").strftime("%d-%m-%Y")

chartdate = noteevents_df['CHARTDATE'].tail(1)
    .to_string(index=False)
chartdate = datetime.strptime(chartdate,
    "%Y-%m-%d").strftime("%d-%m-%Y")

num_note = str(noteevents_df.shape[0])
expired_flag = noteevents_df['EXPIRE_FLAG'].head(1)
    .to_string(index=False)
if expired_flag == "1":
    dod = noteevents_df['DOD'].head(1)
        .to_string(index=False)
    dod = datetime.strptime(dod, "%Y-%m-%d %H:%M:%S").strftime("%d-%m-%Y")
```

Tabella 8.1: Informazioni demografiche e cliniche del paziente

8.2.1.2 Risultati

In Figura sottostante, si osservano informazioni sulla storia clinica del paziente selezionato, comprendendo informazioni come il **nome**, il **numero di identificazione**, la **data di nascita**, l'**ultimo ricovero ospedaliero**, il **numero di note cliniche presenti** e la **data di decesso** (se presente). In questi risultati si evince il nome del paziente, la data di nascita, di ultima ammissione ospedaliera avvenuta l'8 aprile 2148. Inoltre, sono disponibili tre note cliniche relative al paziente. In questo caso, è presente anche la data di morte, deceduto proprio nella stessa data dell'ultimo ricovero ospedaliero, l'8 aprile 2148.

⌚ DASHBOARD OF

Robert Beasley ♂

Patient n°: 27431

📅 Date of birth

04-10-2070

>Last hospital admission

08-04-2148

📝 Number of clinical notes present

3

Date of death

08-04-2148

Figura 8.2: Informazioni demografiche e cliniche del paziente

8.2.2 Estrazione Clinical Trend

La seconda Analytic ha come obiettivo quello di estrarre e visualizzare la tendenza clinica relativa al paziente selezionato. La tendenza clinica fornisce informazioni sull'evoluzione delle condizioni di salute del paziente nel corso del tempo.

8.2.2.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
# Extraction Clinical Trend
clinical_trend = summaries_df[summaries_df['SUBJECT_ID'] ==
    paziente_id]['CLINICAL TREND']
clinical_trend_text = ' '.join(clinical_trend.tolist())

if clinical_trend_text in ['Improvement']:
    delta = 1
elif clinical_trend_text in ['Worsening', 'Dead']:
    delta = -1
else:
    delta = 0 # Se il valore non corrisponde a nessuna
              # delle condizioni precedenti
```

Tabella 8.2: Estrazione Clinical Trend

8.2.2.2 Risultati

In Figura sottostante, si possono visionare sulla dashboard le informazioni riguardante il **Clinical Trend** del paziente. In particolare, emerge lo stato di salute del paziente, in questo caso indicato come "Dead" (-1). Questo dato dunque suggerisce, come anticipato in precedenza, che il paziente in questione è deceduto.

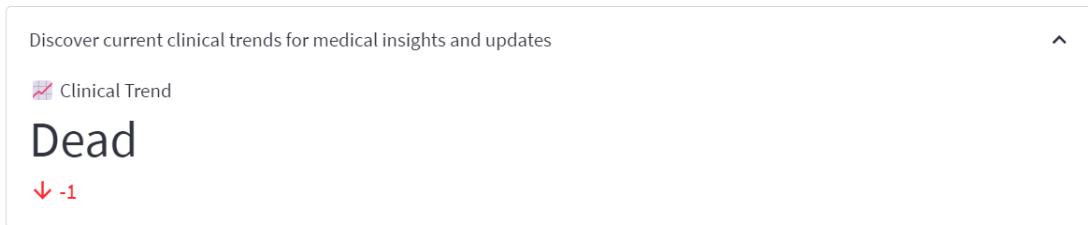


Figura 8.3: Clinical Trend

8.2.3 Estrazione Summary

La terza Analytic ha come obiettivo quello di estrarre e visualizzare un **summary** (ottenuto nella fase di Summarization) che racchiuda le informazioni, sotto forma di testo riassuntivo del quadro clinico, del paziente selezionato. Il summary può essere utile per un quadro riassuntivo che racchiuda le principali condizioni mediche del paziente stesso.

8.2.3.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
# Extraction Summary
summary = summaries_df[summaries_df['SUBJECT_ID'] ==
                      paziente_id]['SUMMARY']
summary_text = ' '.join(summary.tolist())
```

Tabella 8.3: Estrazione Summary

8.2.3.2 Risultati

In Figura sottostante, è possibile osservare sulla dashboard il **summary** ottenuto al seguito dell'esecuzione dell'implementazione precedente. Vengono dunque descritte in dettaglio le informazioni di un paziente di 77 anni, con anamnesi di schizofrenia, CAD, HTN e demenza. Il paziente presentava ipossia e ipotensione, a cui sono seguite intubazione e CVL e, nonostante l'uso di antibiotici, è persistita l'ipossia fino ad un arresto PEA fatale.

The patient is a 77-year-old male nursing home resident with a medical history of schizophrenia, CAD, HTN, and dementia. He presented with hypoxia and hypotension, which led to intubation and CVL placement. He had a recent admission for similar symptoms and was found to have osteomyelitis. The patient also had a history of aspiration pneumonia and chronic decubitus ulcers. Despite treatment with antibiotics, he remained persistently hypoxic and suffered PEA arrest leading to death.

Figura 8.4: Summary

8.2.4 Numero di malattie/sintomi diagnosticati nel tempo

L'obiettivo della quarta Analytic è quello di analizzare il numero delle malattie e dei sintomi diagnosticati nel tempo attraverso le note cliniche del paziente selezionato e di tracciarne l'andamento di questi nel corso del tempo per fornire una comprensione visiva della loro evoluzione.

8.2.4.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
count_note = []
for index, row in noteevents_df.iterrows():
    note = row['ROW_ID']
    date = row['CHARTDATE']
    result = graph_re.run(f"""
        MATCH (p:Patient)-[]-(n:`Note
            Clinical`)-[]-(disease:`Disease or Syndrome`)
        WHERE p.subject_id = {paziente_id} AND
            n.note_clinical_id = {note}
        RETURN count(disease) AS totalOccurrences """)

    for record in result:
        disease_count = str(record)

    result = graph_re.run(f"""
        MATCH (p:Patient)-[]-(n:`Note
            Clinical`)-[]-(disease:`Disease or
            Syndrome`)-[]-(s:`Sign or Symptom`)
        WHERE p.subject_id = {paziente_id} AND
            n.note_clinical_id = {note}
        RETURN count(s) AS totalOccurrences """)

    for record in result:
        symptom_count = str(record)

    count_note.append((note,date,
        disease_count,symptom_count))
```

Tabella 8.4: Numero di malattie/sintomi diagnosticati nel tempo

8.2.4.2 Risultati

In Figura sottostante, è possibile osservare sulla dashboard il **numero di sintomi e malattie** presenti nell'arco temporale che corre tra il 7 Marzo 2148 fino al 7 Aprile 2148 (giorno prima della data di morte). I picchi sia di sintomi che di malattie si hanno avuti il giorno 29 Marzo, in cui si contano ben 36 sintomi e 13 malattie.

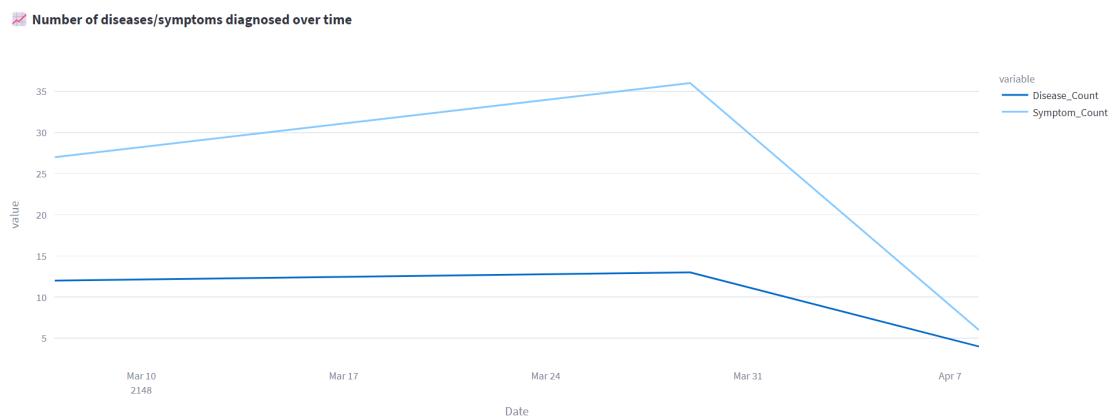


Figura 8.5: Numero di malattie/sintomi diagnosticati nel tempo

8.2.5 Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo

L'obiettivo della quinta Analytic è quello di analizzare il numero delle procedure terapeutiche e di laboratorio eseguite nel tempo attraverso le note cliniche del paziente selezionato e di tracciarne l'andamento di questi nel corso del tempo per fornire una comprensione visiva della loro evoluzione.

8.2.5.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
for index, row in noteevents_df.iterrows():
    note = row['ROW_ID']
    date = row['CHARTDATE']
    result = graph.run(f"""
        MATCH (p:Patient)-[]-(n:`Note
        Clinical`)-[]-(t:`Therapeutic or Preventive
        Procedure`)
        WHERE p.subject_id = {paziente_id} AND
            n.note_clinical_id = {note}
        RETURN count(t) AS totalOccurrences
    """)

    for record in result:
        therapeutic_count = str(record)

    result = graph.run(f"""
        MATCH (p:Patient)-[]-(n:`Note
        Clinical`)-[]-(l:`Laboratory Procedure`)
        WHERE p.subject_id = {paziente_id} AND
            n.note_clinical_id = {note}
        RETURN count(l) AS totalOccurrences
    """)

    for record in result:
        laboratory_count = str(record)
```

Tabella 8.5: Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo

8.2.5.2 Risultati

In Figura sottostante, è possibile osservare sulla dashboard il **numero di procedure terapeutiche/di laboratorio** effettuate nell'arco temporale che corre tra il 3 Marzo 2148 fino al 7 Aprile 2148 (giorno prima della data di morte). I picchi di conteggio del numero di procedure terapeutiche diagnosticate lo si ha il 29 Marzo, con ben 8 procedure. Mentre il picco di conteggio del numero di procedure di laboratorio diagnosticate lo si ha avuto il giorno 7 Marzo, in cui si contano 4 di esse.

Number of therapeutic/laboratory procedure diagnosed over time

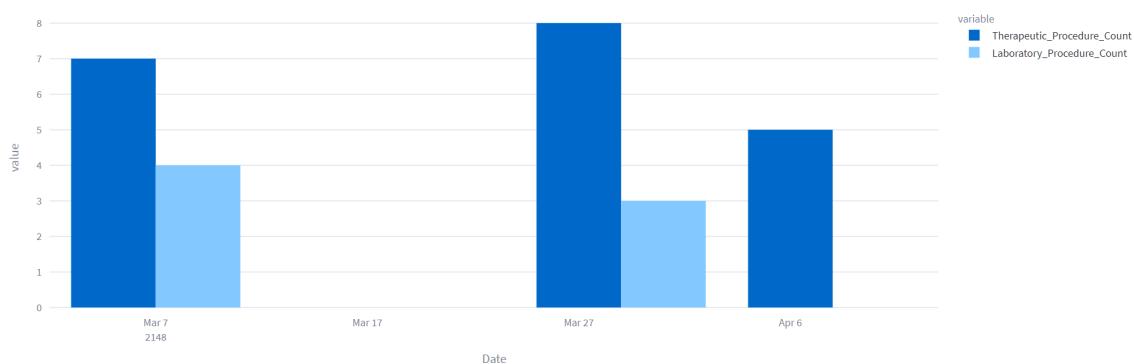


Figura 8.6: Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo

8.2.6 Elenco di farmaci, sintomi e diagnostiche

L'obiettivo della sesta Analytic è quello di estrarre e visualizzare le liste di farmaci, sintomi e procedure diagnostiche correlate alle note cliniche del paziente selezionato, fornendo una panoramica delle sostanze farmacologiche, dei segni/sintomi e delle procedure diagnostiche rilevanti per il paziente.

8.2.6.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result_pharm = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
Syndrome`)-[]-(ps:`Pharmacologic Substance`)
WHERE p.subject_id = {paziente_id} AND ps.name <> " " AND
NOT (toLowerCase(ps.name) CONTAINS "nan")
return DISTINCT(ps) AS Drugs """)

result_symp = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
Syndrome`)-[]-(s:`Sign or Symptom`)
WHERE p.subject_id = {paziente_id} AND s.name <> " " AND
NOT (toLowerCase(s.name) CONTAINS "nan")
return DISTINCT(s) AS Symptoms """)

result_proc = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
Syndrome`)-[]-(dp:`Diagnostic Procedure`)
WHERE p.subject_id = {paziente_id} AND dp.name <> " " AND
NOT (toLowerCase(dp.name) CONTAINS "nan")
return DISTINCT(dp) AS Symptoms """)
```

Tabella 8.6: Elenco di farmaci, sintomi e diagnostiche

8.2.6.2 Risultati

In Figura sottostante, si possono osservare i tre elenchi precedentemente enunciati, che rappresentano delle indicazioni dei **farmaci somministrati**, dei **sintomi presentati** e delle **procedure diagnostiche eseguite** durante il percorso clinico del paziente. Vengono indicati una serie di farmaci utilizzati nel trattamento del paziente, tra cui l'ossigenoterapia, l'ossicodone e l'etomidato. Inoltre, vengono identificati diversi segni o sintomi riportati, come il dolore al torace, le vertigini e la dispnea. Infine, vengono menzionate diverse procedure diagnostiche, tra cui la tomografia computerizzata a raggi X, la radiografia toracica semplice e l'elettrocardiografia.

🔗 💡 Explore list drugs, symptoms, and diagnostics of Robert Beasley

Explore extracted drug, symptom, and diagnostic procedure lists from patient's clinical notes:

| 💊 List of Drugs | ✳️ List of Sign or Symptoms | 🖨️ List of Diagnostic Procedure |
|-----------------|-----------------------------|--------------------------------------|
| Drugs | Symptoms | Diagnostic Procedure |
| Oxygen therapy | Chest Pain | X-Ray Computed Tomography |
| Oxycodone | Dizziness | Plain chest X-ray |
| Etomidate | Dyspnea | Electrocardiography |
| digoxin | Fever | Urinalysis |
| NAC | Dysarthria | Patient Health Questionnaire (PHQ-9) |
| Citalopram | Memory loss | Pulse oximetry |
| Antidepressants | Fatigue | Endoscopy |
| Medical | Headache | Angiogram |
| morphine | Focal neurological deficit | Diagnosis |
| Donepezil | Bone pain | Echocardiography |

Figura 8.7: Elenco di farmaci, sintomi e diagnostiche

8.2.7 Patologie del paziente associate a note cliniche

L'obiettivo della settima Analytic è quello di analizzare le malattie associate alle note cliniche dei pazienti selezionati, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche e malattie per identificare queste ultime (associate a ciascuna nota clinica).

8.2.7.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result1 = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
Syndrome`)
WHERE p.subject_id = {paziente_id}
RETURN p,n,d """)
```

Tabella 8.7: Patologie del paziente associate a note cliniche

8.2.7.2 Risultati

In Figura sottostante sono mostrate le **patologie** associate alla nota clinica di un paziente selezionato, si può notare che le malattie/sindromi sono associate a più cartelle cliniche, evidenziando una forte presenza di quella particolare malattia nella storia del paziente. Questa osservazione potrebbe suggerire ai medici di monitorare attentamente e trattare adeguatamente tali condizioni nel contesto della cura del paziente selezionato.

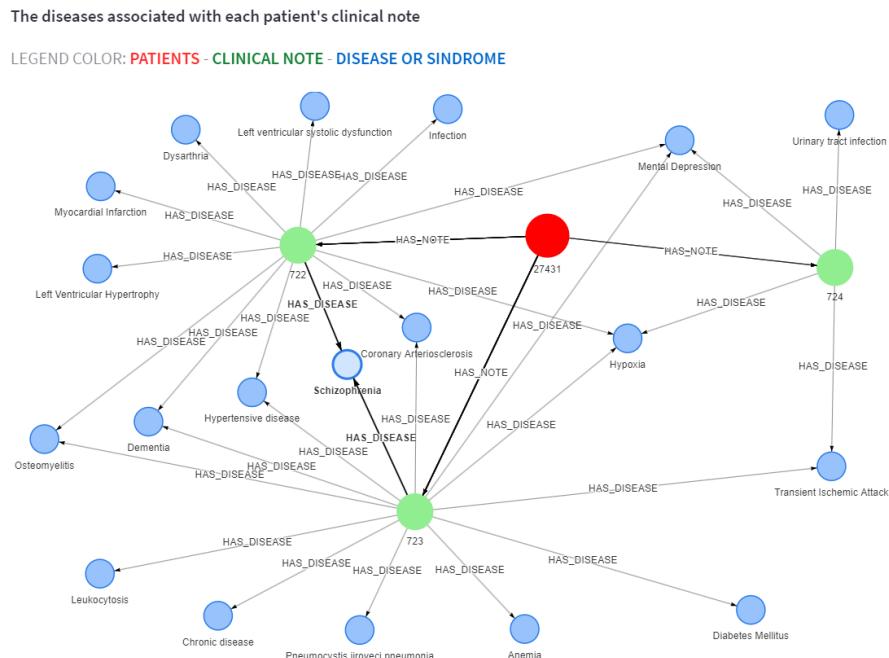


Figura 8.8: Patologie del paziente associate a note cliniche

8.2.8 Parti del corpo doloranti del paziente associate alle note cliniche

L'obiettivo dell'ottava Analytic è quello di analizzare e visualizzare le parti del corpo doloranti associate alle note cliniche dei pazienti selezionati, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche e parti del corpo doloranti per identificare queste ultime in ciascuna nota clinica.

8.2.8.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result2 = graph.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]->(b:`Body Part`)
WHERE p.subject_id = {paziente_id}
RETURN p,b,n """)
```

Tabella 8.8: Parti del corpo doloranti del paziente associate alle note cliniche

8.2.8.2 Risultati

In Figura sottostante vengono mostrate le **parti del corpo** per la quale si prova dolore, chiaramente associate alla nota clinica di un paziente selezionato. Come si può evincere, al paziente con identificativo 27431 sono associate tre cartelle cliniche, ognuna delle quali mostra le parti del corpo ad essa associata. Per la cartella clinica 722 sono presenti più parti del corpo doloranti, tra cui la struttura della parete cardiaca, la struttura dell'atrio destro, la struttura del lobo temporale destro, cervello e setto cardiaco.

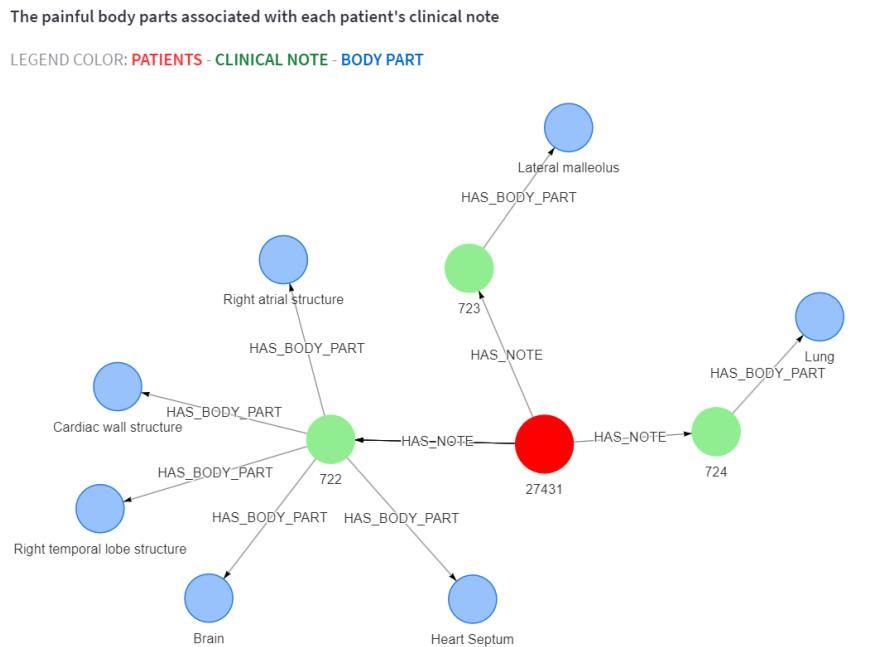


Figura 8.9: Parti del corpo doloranti del paziente associate alle note cliniche

8.2.9 Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche

L'obiettivo della nona Analytic è quello di analizzare e visualizzare le sostanze farmacologiche prese dal paziente per ciascuna malattia diagnosticata nelle sue note cliniche, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche, malattie e sostanze farmacologiche per identificare i trattamenti di specifiche malattie.

8.2.9.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result3 = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
    Syndrome`)-[]-(ps:`Pharmacologic Substance`)
WHERE p.subject_id = {paziente_id}
RETURN p,n,d,ps """)
```

Tabella 8.9: Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche

8.2.9.2 Risultati

In Figura sottostante, si osservano le **sostanze farmacologiche** assunte dal paziente per ciascuna malattia diagnosticata nelle sue note cliniche. Ciò permette di identificare rapidamente l'associazione di specifiche sostanze farmacologiche a singole malattie, indicando una forte correlazione tra l'uso di tali farmaci e la gestione di quelle specifiche condizioni.

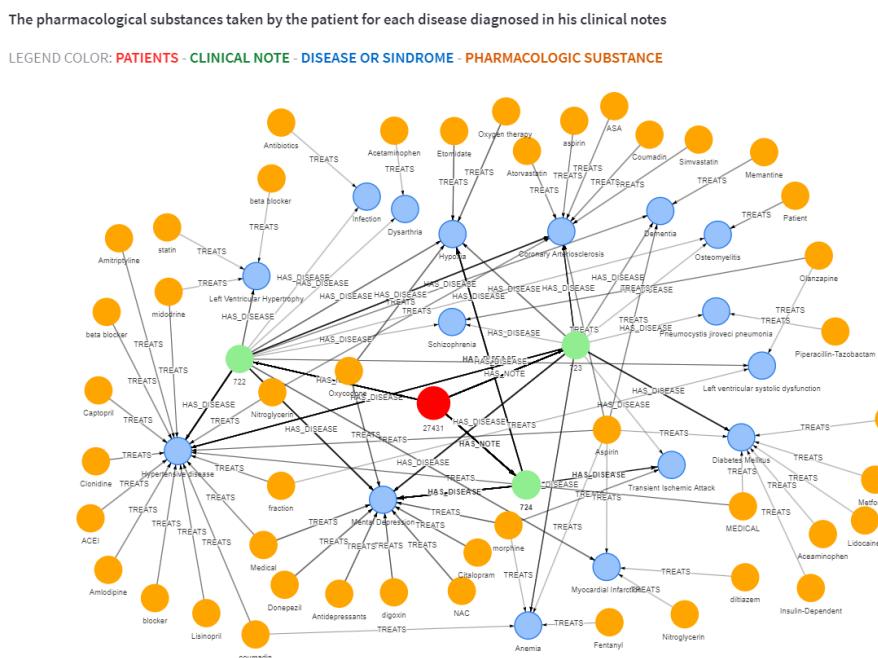


Figura 8.10: Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche

8.2.10 Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche

L'obiettivo della decima Analytic è quello di analizzare e visualizzare i sintomi presentati dal paziente per ciascuna malattia diagnosticata nelle sue note cliniche, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche, malattie e sintomi per identificare i sintomi associati a specifiche malattie.

8.2.10.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result4 = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
    Syndrome`)-[]-(s:`Sign or Symptom`)
WHERE p.subject_id = {paziente_id}
RETURN p,n,d,s """)
```

Tabella 8.10: Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche

8.2.10.2 Risultati

Nella Figura sottostante vengono mostrati i **sintomi** presentati dal paziente per ciascuna malattia diagnosticata nelle sue note cliniche. Questo approccio permette di individuare rapidamente le correlazioni tra i sintomi presentati e le specifiche malattie o condizioni.

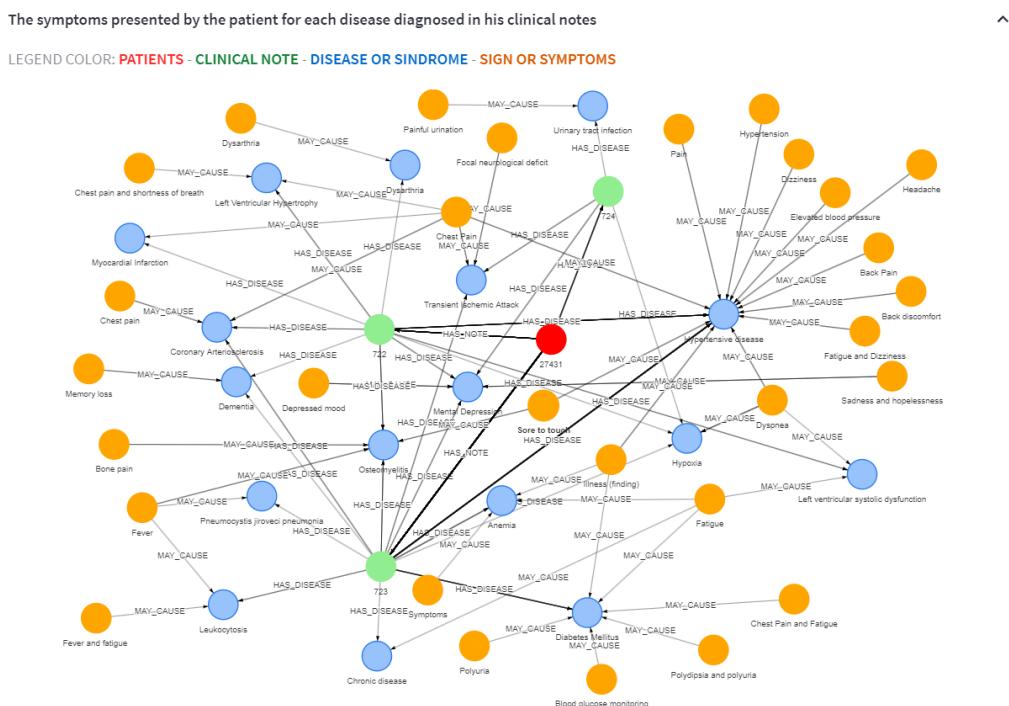


Figura 8.11: Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche

8.2.11 Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche

L'obiettivo dell'undicesima Analytic è quello di analizzare e visualizzare le procedure diagnostiche eseguite sul paziente per ciascuna malattia diagnosticata nelle sue note cliniche, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche, malattie e procedure diagnostiche al fine di identificare le procedure diagnostiche associate a specifiche malattie.

8.2.11.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result5 = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
    Syndrome`)-[]-(dp:`Diagnostic Procedure`)
WHERE p.subject_id = {paziente_id}
RETURN p,n,d,dp """)
```

Tabella 8.11: Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche

8.2.11.2 Risultati

In Figura sottostante si osservano le **procedure diagnostiche** eseguite sul paziente per ciascuna malattia diagnosticata nelle sue note cliniche, questa visualizzazione offre una panoramica delle procedure diagnostiche effettuate per valutare e confermare le diverse condizioni di salute del paziente.

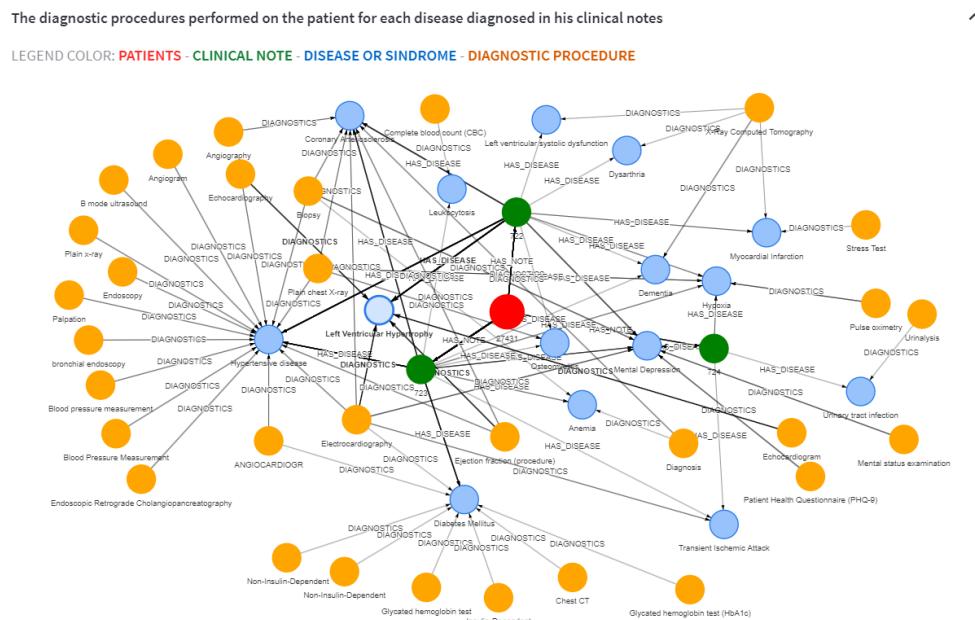


Figura 8.12: Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche

8.2.12 Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche

L'obiettivo della dodicesima Analytic è quello di analizzare e visualizzare i test di laboratorio eseguiti sul paziente per ciascuna malattia diagnosticata nelle sue note cliniche, fornendo una rappresentazione grafica delle relazioni tra pazienti, note cliniche, malattie e test di laboratorio al fine di identificare i test di laboratorio associati a specifiche malattie.

8.2.12.1 Implementazione

Segue l'implementazione scritta in **Python**:

```
result6 = graph_re.run(f"""
MATCH (p:Patient)-[]-(n:`Note Clinical`)-[]-(d:`Disease or
    Syndrome`)-[]-(l:`Laboratory or Test Result`)
WHERE p.subject_id = {paziente_id}
RETURN p,n,d,l """)
```

Tabella 8.12: Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche

8.2.12.2 Risultati

In Figura sottostante si osservano i **test di laboratorio** eseguiti sul paziente per ciascuna malattia diagnosticata nelle sue note cliniche, questa rappresentazione visiva dei dati fornisce un metodo per poter analizzare e valutare l'importanza dei test di laboratorio nel processo diagnostico e nel monitoraggio delle condizioni di un paziente.

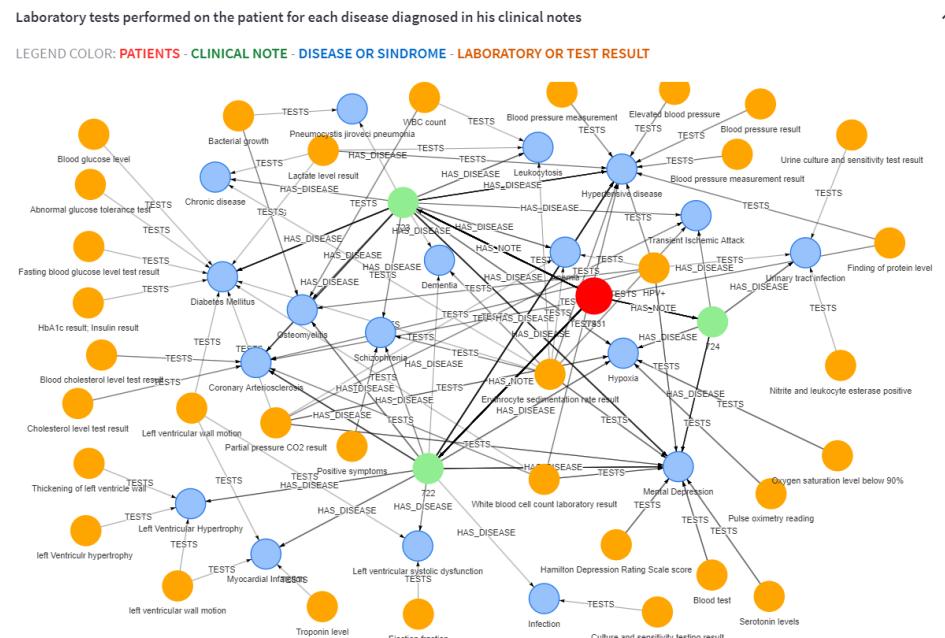


Figura 8.13: Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche

9 Conclusioni

Contenuti

| | | |
|-----|---|----|
| 9.1 | Efficienza e riduzione del carico di lavoro | 73 |
| 9.2 | Miglioramento della qualità dell'assistenza | 73 |
| 9.3 | Supporto decisionale clinico | 73 |
| 9.4 | Costi di realizzazione del progetto effettivi e futuri | 73 |
| 9.5 | Tempi di esecuzione | 75 |
| 9.6 | Sviluppi futuri: Migrazione da MongoDB e Neo4J verso ArangoDB | 76 |
| 9.7 | Materiale del progetto | 76 |

In conclusione si possono trarre diversi scopi finali e scopi futuri dell'Healthcare Summarization.

9.1 Efficienza e riduzione del carico di lavoro

L'implementazione di un sistema di Healthcare Summarization può ridurre il tempo e lo sforzo necessari per l'analisi e la comprensione delle informazioni contenute nelle note cliniche. La generazione di riassunti accurati e pertinenti può semplificare la revisione delle informazioni da parte dei professionisti sanitari, consentendo loro di concentrarsi su aspetti critici dei dati e prendere decisioni informate in modo più efficiente.

9.2 Miglioramento della qualità dell'assistenza

Un progetto di Healthcare Summarization può contribuire a migliorare la qualità dell'assistenza fornita ai pazienti. Estrarre informazioni chiave, come diagnosi, terapie e condizioni mediche, consente ai fornitori di assistenza sanitaria di ottenere una visione più completa del quadro clinico dei pazienti. Ciò può facilitare una diagnosi più accurata, un trattamento appropriato e un monitoraggio efficace delle condizioni dei pazienti.

9.3 Supporto decisionale clinico

L'estrazione e la sintesi delle informazioni da fonti sanitarie possono fornire un supporto decisionale più solido per i professionisti sanitari. Riassumere i dati chiave, come i risultati dei test di laboratorio, le condizioni pregresse dei pazienti o le interazioni tra farmaci, può aiutare i medici a prendere decisioni informate e adottare piani di cura personalizzati.

9.4 Costi di realizzazione del progetto effettivi e futuri

Si elencano i costi di realizzazione del progetto associati all'utilizzo di infrastrutture tecnologiche e all'utilizzo di modelli di OpenAI usati.

● Costi effettivi

- **Acquisizione e gestione dei dati:** - *FREE* - La raccolta dei dati da MIMIC III ha un processo di assicurazione sull'accesso ove è richiesto un procedimento lungo e controllato ma tutto gratuito.
- **Infrastruttura Tecnologica**

- * **MongoDB** - *FREE* - Accesso alle risorse e archiviazione limitato;
- * **Neo4J** - *FREE* - Servizio usato solamente in locale.
- **Modelli di Open AI** - \$12.21 - Accesso alla API KEY e utilizzo del modello GPT-3.5-Turbo con i seguenti limiti:
 - * Input - Output token totali: 4097
 - * \$0.002/1K Tokens per uso completo (Prompt + Completion)

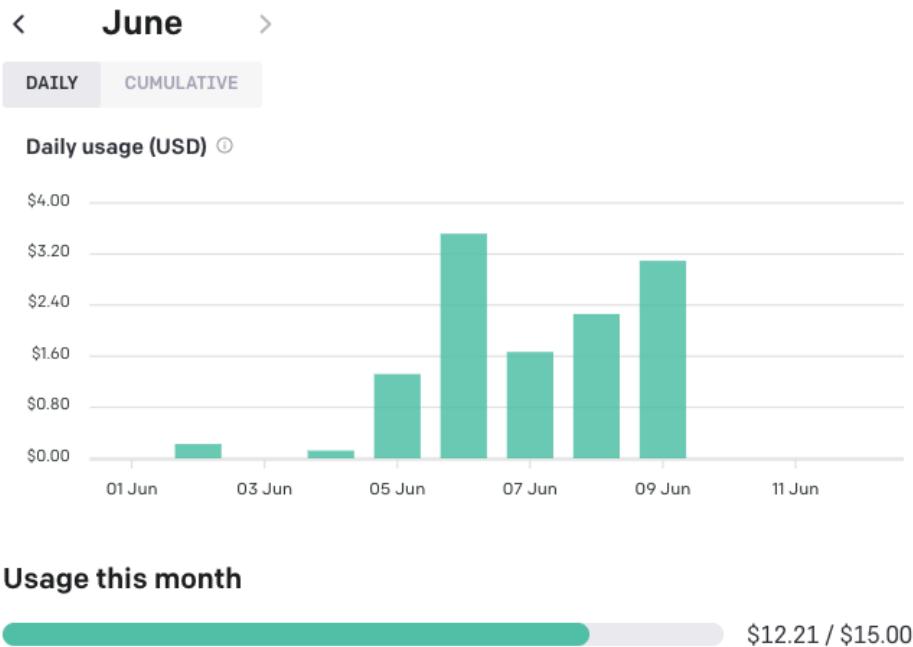


Figura 9.1: Costi effettuati nel mese di Giugno

● Costi futuri

- **Acquisizione e gestione dei dati**: - *FREE* - La raccolta dei dati da MIMIC III ha un processo di assicurazione sull'accesso ove è richiesto un procedimento lungo e controllato ma tutto gratuito.
- **Infrastruttura Tecnologica**
 - * **MongoDB** - \$ 57/Mese - Accesso alle risorse e archiviazione illimitato, rendendo il servizio completamente in cloud e super scalabile;
 - * **Neo4J** - \$ 65/Mese - AuraDB Neo4J
- **Modelli di Open AI** - \$0.06 / 1K tokens + \$0.12 / 1K tokens - Accesso alla API KEY e utilizzo del modello GPT-4 con i seguenti limiti:
 - * Input - Output token totali: 32K

9.5 Tempi di esecuzione

Si è deciso di effettuare un'**analisi quantitativa** dei tempi di esecuzione al variare del numero di pazienti selezionati¹:

| Operazione | Numero di pazienti selezionati | | |
|---------------------|--------------------------------|-----|------|
| | 20 | 50 | 75 |
| Retrieve | 7 | 16 | 27 |
| Preprocessing | 129 | 273 | 424 |
| Summarization | 298 | 617 | 1037 |
| NER | 63 | 131 | 198 |
| Relation Extraction | 913 | - | - |
| Graph Modeling | 55 | 104 | 157 |

Tabella 9.1: Analisi di scalabilità - Tempi di esecuzione

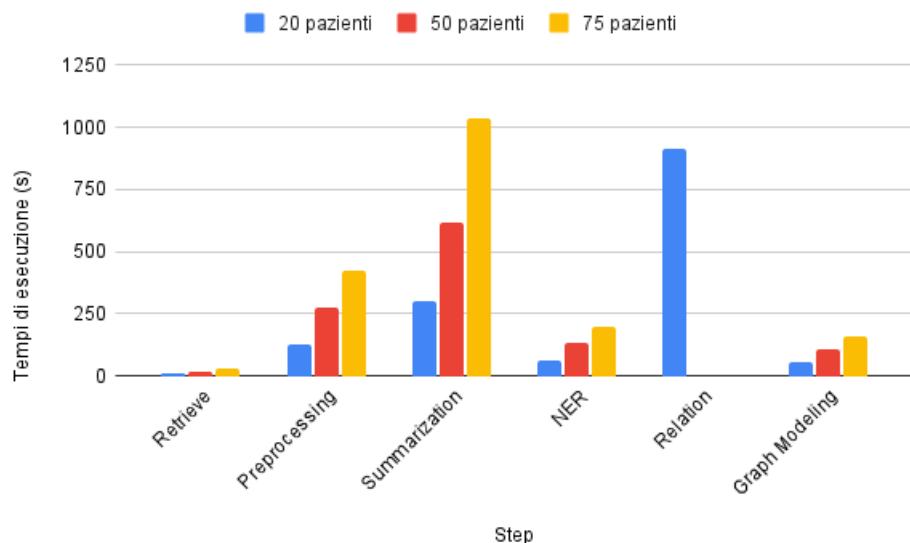


Figura 9.2: Analisi di Scalabilità - Tempi di esecuzione

Si noti: che i tempi di esecuzione precedenti sono ottenuti a valle di una stima valutativa ottenuta sull'esperienza effettuata durante lo svolgimento delle suddette operazioni.

¹ Per l'operazione di Relation Extraction si è omesso di effettuare l'analisi per 50 e 75 in quanto i tempi di esecuzione risultavano già molto onerosi per 20 pazienti.

9.6 Sviluppi futuri: Migrazione da MongoDB e Neo4J verso ArangoDB

In questo progetto, si è utilizzata un'architettura basata su MongoDB e Neo4J, tuttavia, come possibile sviluppo futuro del progetto si è pensato ad una migrazione verso l'architettura di **ArangoDB**. Questa scelta risulterebbe vantaggiosa sia per la sua scalabilità, flessibilità del modello dei dati sia per la sua forte integrazione con lo stack tecnologico esistente.

Infatti, ArangoDB ha il grande vantaggio di gestire efficacemente **sia i dati documentali che le relazioni complesse su grafo**. La flessibilità del modello dei dati consente di modellare i dati in modo più accurato e intuitivo, semplificando le operazioni di query complesse.

In conclusione, **il passaggio da MongoDB e Neo4J verso ArangoDB potrebbe fornire una soluzione potente e scalabile per la gestione dei dati in esame, consentendo di ottimizzare le prestazioni e migliorare l'efficienza complessiva del sistema di Healthcare Summarization**.

9.7 Materiale del progetto

Per il materiale completo mostrato nel presente elaborato, si rimanda il lettore al repository ufficiale del progetto contenente tutti i notebook **Python** nonché un **video dimostrativo** della dashboard in esecuzione:

[GitHub](#)

<https://github.com/giuseppericcio/HealthcareSummarizationMIMIC>

Elenco delle figure

| | |
|---|----|
| 1.1 Workflow del progetto | 4 |
| 2.1 Plot - Distribuzione Documenti per Categoria | 6 |
| 2.2 Plot - Distribuzione Documenti per Lunghezza Carattere | 7 |
| 2.3 Plot - Numero Documenti per Paziente | 8 |
| 2.4 Plot - Distribuzione Documenti per Lunghezza Carattere | 11 |
| 2.5 Plot - Numero Documenti per Paziente | 12 |
| 3.1 Architettura di Memorizzazione | 18 |
| 5.1 Architettura Clinical Notes a Summary e Clinical Trend | 26 |
| 6.1 Architettura NER + EL | 33 |
| 6.2 Recognized Entities | 38 |
| 6.3 Workflow NER + EL | 40 |
| 6.4 Architettura RE | 41 |
| 6.5 Architettura NER+EL e RE | 45 |
| 7.1 Dalla NER+EL+RE a Neo4J | 46 |
| 7.2 Concept Graph Schema | 47 |
| 7.3 Concept Graph Network | 48 |
| 7.4 Knowledge Graph Schema | 49 |
| 7.5 Knowledge Graph Network | 50 |
| 7.6 Entità cliniche collegate alla nota clinica | 53 |
| 7.7 Relazione tra entità cliniche | 54 |
| 8.1 Dashboard | 56 |
| 8.2 Informazioni demografiche e cliniche del paziente | 58 |
| 8.3 Clinical Trend | 59 |
| 8.4 Summary | 60 |
| 8.5 Numero di malattie/sintomi diagnosticati nel tempo | 62 |
| 8.6 Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo | 64 |
| 8.7 Elenco di farmaci, sintomi e diagnostiche | 66 |
| 8.8 Patologie del paziente associate a note cliniche | 67 |
| 8.9 Parti del corpo doloranti del paziente associate alle note cliniche | 68 |
| 8.10 Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche | 69 |
| 8.11 Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche | 70 |
| 8.12 Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche | 71 |
| 8.13 Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche | 72 |
| 9.1 Costi effettuati nel mese di Giugno | 74 |
| 9.2 Analisi di Scalabilità - Tempi di esecuzione | 75 |

Elenco delle tabelle

| | |
|---|----|
| 2.1 Dataframe estratto da Note Events | 9 |
| 2.2 Dataframe estratto da Note Events con Token Count | 10 |
| 2.3 Dataframe estratto da Patients | 14 |
| 2.4 Dataframe finale estratto da Note Events e Patients | 14 |
| 3.1 Workflow: Data Collection - Python - MongoDB Atlas | 17 |
| 4.1 Codice 1: Section extraction | 24 |
| 4.2 Codice 2: Lemmatization | 25 |
| 5.1 Scelta dei modelli Open AI | 26 |
| 6.1 Metriche di valutazione MedCAT | 32 |
| 6.2 Annotazione su testo preprocessato | 37 |
| 7.1 Graph Modeling - Cypher – Neo4j | 53 |
| 8.1 Informazioni demografiche e cliniche del paziente | 57 |
| 8.2 Estrazione Clinical Trend | 59 |
| 8.3 Estrazione Summary | 60 |
| 8.4 Numero di malattie/sintomi diagnosticati nel tempo | 61 |
| 8.5 Numero di procedure terapeutiche/di laboratorio diagnosticate nel tempo | 63 |
| 8.6 Elenco di farmaci, sintomi e diagnostiche | 65 |
| 8.7 Patologie del paziente associate a note cliniche | 67 |
| 8.8 Parti del corpo doloranti del paziente associate alle note cliniche | 68 |
| 8.9 Sostanze farmacologiche assunte dal paziente per ogni patologia diagnosticata nelle note cliniche | 69 |
| 8.10 Sintomi presentati dal paziente per ogni malattia diagnosticata nelle note cliniche | 70 |
| 8.11 Procedure diagnostiche eseguite sul paziente per ogni malattia diagnosticata nelle note cliniche | 71 |
| 8.12 Esami di laboratorio eseguiti sul paziente per ogni malattia diagnosticata nelle sue note cliniche | 72 |
| 9.1 Analisi di scalabilità - Tempi di esecuzione | 75 |

Bibliografia

- [1] David Charles et al. *Adoption of Electronic Health record Systems among U.S. Non-federal Acute Care Hospitals*. ONC Data Brief No. 9, 1-9 (2013). 2013.
- [2] Francis S. Collins e Lawrence A. Tabak. *NIH plans to enhance reproducibility*. Nature 505, 612-613 (2014). 2014.
- [3] H. Eyre et al. *Launching into clinical space with medspaCy: a new clinical text processing toolkit in Python*. AMIA Annu Symp Proc, 438-447 (2021). 2021.
- [4] Zeljko Kraljevic. *Exploring Electronic Health Records with MedCAT and Neo4j*. <https://towardsdatascience.com/exploring-electronic-health-records-with-medcat-and-neo4j-f376c03d8eef>. 2021.
- [5] Zeljko Kraljevic et al. *Multi-domain clinical natural language processing with MedCAT: The Medical Concept Annotation Toolkit*. Artif. Intell. Med. vol. 117. 2021.
- [6] Sunil Mohan e Donghui Li. *MedMentions: A Large Biomedical Corpus Annotated with UMLS Concepts*. In *Proceedings of the 2019 Conference on Automated Knowledge Base Construction (AKBC 2019)*. 2019.
- [7] PhysioNet. *MIMIC-III Clinical Database*. <https://physionet.org/content/mimiciii/1.4/>. 2016.
- [8] Thomas Searle et al. *Discharge summary hospital course summarisation of in patient Electronic Health Record text with clinical concept guided deep pre-trained Transformer models*. <https://www.sciencedirect.com/science/article/abs/pii/S1532046423000795>. 2023.