UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"ANALISI TESTUALE COMPUTAZIONALE: IL CASO STUDIO DELLE OPERE DI SHAKESPEARE"

Relatore: Prof. Di Nunzio Giorgio Maria

Laureando: Calciolari Andrea

ANNO ACCADEMICO 2022 –2023

Data di laurea 16/11/2023

Abstract

L'obbiettivo della tesi è la progettazione di codice Python per la visualizzazione e analisi di un database contenente informazioni riguardanti una delle maggiori opere di William Shakespeare, "Otello", utilizzando il framework Shiny. Vengono discusse le scelte implementative riguardanti la gestione dei dati e la realizzazione di un'interfaccia utente che permetta una visualizzazione dinamica e interattiva dei dati presi in esame, prestando particolare attenzione alla distribuzione dei diversi valori di *Logic* e *Semantics* all'interno del database.

Indice

Introduzione

Durante il processo di sviluppo di un database, ci sono diversi passaggi fondamentali che devono esser fatti per giungere a un prodotto finale completo e funzionale. Questo percorso inizia dalla raccolta dei dati per poi proseguire con la loro organizzazione in diagrammi relazionali e logici, per consentirne un accesso ordinato e immediato alle informazioni quando necessario. Tuttavia, un aspetto altrettanto cruciale nella creazione di un database è la progettazione di un'interfaccia utente che renda possibile e facile l'estrazione delle informazioni di cui abbiamo bisogno dalla base di dati.

Una User Interface deve essere prima di tutto completa e intuitiva; infatti, essa gioca un ruolo fondamentale nell'offrire agli utenti una visione completa e chiara dei dati analizzati.

La tesi che segue si concentra proprio su questa fase finale del processo. Partendo da un file che analizza la celebre tragedia di William Shakespeare, "Otello", catalogando le interazioni tra i vari personaggi atto per atto e scena per scena, si vuole arrivare alla creazione di un'interfaccia utente che consenta di visualizzare e interagire con il database in modo efficace.

Per quanto riguarda l'analisi dell'opera, l'obbiettivo sarà quello di estrarre statistiche dettagliate sul modo in cui i diversi aspetti di logica e semantica sono distribuiti al suo interno. Inoltre, l'interfaccia deve anche fornire agli utenti la possibilità di concentrarsi su un personaggio o interazioni specifiche qualora lo desiderano.

Il processo inizierà quindi con un'analisi dettagliata dei dati a disposizione, al fine di comprendere quali sono le operazioni necessarie per gestire i dati in maniera più adatta. Successivamente, verrà creata una tabella visualizzabile dall'utente tramite un'interfaccia appositamente progettata per questo scopo.

Per la realizzazione di questo progetto, abbiamo deciso di utilizzare il linguaggio di programmazione Python come piattaforma principale. La motivazione dietro questa scelta deriva da una serie di vantaggi che questo linguaggio offre rispetto ad altre alternative. Uno di questi benefici è sicuramente l'ampia gamma di librerie e framework che si hanno a disposizione quando si realizza codice con Python. L'utilizzo di queste risorse ci consente di scrivere codice in maniera più efficiente e servendoci di funzioni specializzate.

Alla base della realizzazione dell'applicativo ci sarà infatti Shiny, un framework che consente la realizzazione di applicativi web interattivi utilizzando interamente R o, come nel nostro caso, Python.

Nonostante questo, Shiny offre un ampio grado di personalizzazione e risulta inoltre compatibile con le principali librerie per la gestione e l'analisi di dati che andremo ad utilizzare durante la realizzazione dell'applicativo come Pandas, NumPy e Matplotlib.

2. Analisi del progetto

Prima di addentrarci nella fase di scrittura del codice, è di fondamentale importanza dedicare del tempo all'analisi attenta del nostro database di partenza, che in questo caso è rappresentato da un file *.xlsx*. Questo passaggio preliminare riveste un'importanza cruciale, poiché ci fornisce una visione chiara della struttura dei dati, delle informazioni che contiene e delle relazioni tra gli elementi.

Una volta fatto questo, dovremo affrontare l'importante compito di decidere come gestire le peculiarità del nostro database all'interno del codice. Questo processo richiede una valutazione attenta e un piano strategico per l'implementazione delle operazioni di lettura, scrittura, aggiornamento e manipolazione dei dati. Le decisioni prese in questa fase determineranno la progettazione dell'applicazione che stiamo sviluppando e dovranno essere basate sulla nostra comprensione delle esigenze e dei requisiti specifici del progetto, tenendo conto delle caratteristiche del file da cui stiamo estrapolando i dati.

L'analisi e la gestione del database costituiscono quindi le fondamenta su cui costruire il nostro progetto. Una buona interpretazione dei dati e una conseguente decisione delle scelte implementative saranno determinanti per il successo complessivo del progetto, poiché influenzeranno direttamente la qualità e l'efficacia del codice e dell'applicazione finale.

ACT	SCENE	SPEAKER	ADDRESSEE	LOGIC			SEMANTICS			WORDS								
	1.1	RODERIGO	IAGO	FIRST			MANIFEST			purse	strings	know	follow					
IGO AND FIRSTNESS		IAGO	RODERIGO	ECHO	FOREGONE		MANIFEST			matter	dream	abhor	hear	follow				
V, OLD BLACK RAM, DEVIL		IAGO	RODERIGO	ECHO			MANIFEST	SEEMING		price	worth	place	pride	bombast	nonsuits	mediators		
SIGN		IAGO	RODERIGO	ECHO			MANIFEST	SEEMING		fair ville	spinster	bookish	theoric	prattle	practice	counter-caster		
		IAGO	RODERIGO	FIRST	FOREGONE		MANIFEST	SEEMING		eyes	proof	grounds	debitor	creditor	preferment	letter	allection	old gradation	second
NG THE BLACK INVISIBLE		IAGO	RODERIGO	ECHO			MANIFEST	BLACK		follow	serve	turn	masters	followed	duteous	knee-crooking	knave	bondage	ass
Jwem		IAGO	RODERIGO	ECHO			SEEMING	INVISIBLE		honest knaves	forms	visages	hearts	shows	service	ords	homage	following	him
										outward	action	demonstrate	native act	figure	heart	complement	extern	dawn	
black ram		IAGO	BRABANTIO	FIRST	FOREGONE		BLACK	INVISIBLE		robbed	heart	lost	old black ram	tupping	white ewe	devil		
		IAGO	BRABANTIO	FIRST	FOREGONE		MANIFEST	INVISIBLE	BLACK	God	devil	service	ruffians	Barbary horse	neigh	coursers	jennets	beast with two backs	
		RODERIGO	BRABANTIO	ECHO			INVISIBLE	BLACK		knave	gross clasps	lascivious	Moor	gross	revolt	extravagant	wheeling	stranger	here and everywhere
		IAGO	RODERIGO	ECHO	FIRST		INVISIBLE			hell-pains	necessity	show	flag	sign	Sagittary				
		BRABANTIO	RODERIGO	ECHO			INVISIBLE	SEEMING	BLACK	treason	blood	minds	act	charms	property	abused			
INTRO CONSCIENC	1.2	IAGO	OTHELLO	FIRST	ECHO	FOREGONE	MANIFEST	INVISIBLE		plain	men	stuff	conscience	contrived	murder	lack iniquity	service	prated	
IAGO, murder		IAGO	OTHELLO	FIRST	ECHO	FOREGONE	MANIFEST	INVISIBLE		scurvy	provoking terms	honour	married	divorce	restraint	grievance	law	cable	
manifest		OTHELLO	IAGO	ECHO			MANIFEST			services	out-tongue	complaints	boasting	honour	life men	royal siege	demerits	unbonneted	
OTHELLO honour		OTHELLO	IAGO	ECHO			MANIFEST			proud	fortune	unhoused	free	condition	circumscription	confine	sea's worth		
										parts	title	perfect	soul	manifest					
stealing		BRABANTIO	OTHELLO	ECHO			BLACK	INVISIBLE		thief	enchanted	things of sense	chains of magic	bound	tender	fair	happy		
blackness		BRABANTIO	OTHELLO	ECHO			BLACK	INVISIBLE		sooty bosom	thing	gross in sense	foul charms	drugs	minerals	palpable	abuser	practicer	arts inhibited
Senate	1.3	DUKE	SENATOR 1	FIRST			SEEMING			no composition	disproportioned	difference	error	pageant	false gaze				
		BRABANTIO	DUKE	ECHO			SEEMING			flood-gate	o'erbearing	ingluts	swallows	abused	stolen	corrupted	spells	medicines	mountebanks
rabantio: error of nature		BRABANTIO	DUKE	ECHO	FOREGONE		SEEMING			nature preposterous	err	deficient	blind	lame of sense	witchcraft				
		OTHELLO	DUKE	ECHO			MANIFEST			true	head	front	rude	soft phrase	arms	action	broil	battle	round unvarnished
introducing		BRABANTIO	DUKE	ECHO			BLACK			still	quiet	nature	feared	judgement	maimed	imperfect	err		
witchcraft		BRABANTIO	DUKE	ECHO			BLACK	INVISIBLE		rules of nature	practices	cunning	hell	mixtures	blood	dram	conjured	effect	
OTHELLO'S TALE																			
echoing		OTHELLO	DUKE	ECHO	FIRST		MANIFEST	BLACK		vices	blood	grave ears	ran it through	t disastrous chairmoving accidents		flood	field	hair-breadth escapes	imminent

Figura 2.1.1: Sezione dei dati presi in esame

In questo progetto di tesi, il file su cui lavoreremo consiste in una tabella contenente una rappresentazione dei dialoghi presenti all'interno di "Otello", la famosa tragedia scritta da William Shakespeare.
A ciascuna interazione vengono attribuiti specifici valori secondo diversi parametri, delineando in questo modo gli elementi chiave del dialogo. Questi attributi includono:

- **Speaker**: Identifica il personaggio che sta attualmente parlando.
- **Addressee**: Indica il personaggio a cui lo Speaker si sta rivolgendo.
- **Logic**: Riflette il tipo di logica presente nel dialogo e riguarda quindi la struttura razionale delle idee e la coerenza nell'esposizione degli argomenti.
- **Semantics**: Indica la semantica utilizzata il significato, che serve a determinare il significato delle frasi all'interno dell'opera.
- **Words**: Identifica le parole chiave che usate all'interno del dialogo.

Attraverso l'assegnazione di valori a ciascuno di questi attributi, si mira a fornire una descrizione completa di tutte le interazioni contenute all'interno della tragedia.
Una volta individuati gli attributi di ogni dialogo, possiamo compiere un'ulteriore analisi per determinare quali sono le caratteristiche del file che potrebbero causare problemi durante la realizzazione del codice. Le principali sono:

- **Celle vuote**: Il database presenta diverse righe e colonne vuote; sarà importante considerare come gestire queste celle vuote durante la fase di analisi ed elaborazione, poiché potrebbero rappresentare dati mancanti o non rilevanti.
- **Struttura delle colonne**: Alcuni attributi, come *Logic*, *Semantics*, e *Words*, presentano valori che occupano più colonne, nonostante l'intestazione (*header*) sia presente solo sulla prima colonna del gruppo. Questa struttura potrebbe richiedere un'elaborazione speciale per estrarre e interpretare correttamente i valori di questi campi.
- **Termini evidenziati**: Nel campo *Words* alcuni termini sono evidenziati, ma questa caratteristica risulta irrilevante per quanto riguarda l'obbiettivo finale dell'applicazione. Sceglieremo quindi di tralasciare questa caratteristica, prediligendo un mantenimento della leggibilità della tabella.
- **Entry incomplete**: Nel database sono presenti voci incomplete, ossia che mancano di alcuni attributi. Sarà importante gestirle in maniera adeguata per evitare anomalie durante la raccolta dei dati per la realizzazione di statistiche.

Sarà importante assicurarsi che la gestione di questi aspetti sia conforme agli obbiettivi che sono stati posti e che venga rispecchiata correttamente all'interno del codice.

3. Implementazione

3.1 Librerie per la realizzazione dell'applicativo

Per questo progetto useremo alcune librerie di Python;

1. **Pandas**: nome che deriva da "Panel Data" e spesso nota semplicemente come "pd", è una libreria utilizzata principalmente per la lettura e l'analisi dei dati, oltre che la creazione di strutture dati utili per rappresentare database e tabelle. Una di queste strutture è il DataFrame che permette di organizzare i dati in formato tabulare, e di applicare operazione logiche e matematiche lungo righe e colonne.
2. **Matplotlib:** una libreria che vede ampio utilizzo in ambito scientifico poiché permette di rappresentare attraverso diversi tipi di grafici le informazioni riguardanti una particolare set di dati.
3. **Shiny:** un framework nato inizialmente per creare applicativi web interattivi utilizzando R come linguaggio di programmazione, ma che è successivamente stato aggiornato per essere compatibile anche con Python. Grazie alle funzionalità di Shiny, è quindi possibile generare una pagina web che può venire modificata interagendo con un'interfaccia utente realizzata appositamente.

3.2 Lettura e manipolazione del database

Prima di poter iniziare le funzionalità di Shiny, dobbiamo però ottenere, a partire dal nostro database, un oggetto che sia compatibile con i metodi della libreria e che non presenti i problemi discussi in precedenza.

Utilizzando Pandas, leggiamo il database dal file *.xlsx* chiamando la funzione *pd.read_excel(),* che legge il file passato come parametro e restituisce un oggetto di tipo Dataframe; i valori contenuti nella prima riga verranno utilizzati come intestazione di ciascuna delle colonne.

Procediamo poi con il rimuovere le righe e le colonne composte unicamente da celle vuote, invocando *pd.DataFrame.dropna()* per rimuovere le righe (*axis=0*) e le colonne (*axis=1*) dove tutte le celle sono vuote.

```
df = pd.read_excel('othello.xlsx', sheet_name=1)
df = pd.DataFrame.dropna(df, axis=0, how='all')
df = pd.DataFrame.dropna(df, axis=1, how='all')
```

Figura 3.2.1: Codice usato per la lettura del file

Successivamente, rimuoviamo anche le righe che hanno valore di *Speaker*, *Addressee*, *Logic* o *Semantics* nullo, poiché costituiscono entry incomplete e causerebbero imprecisioni in fase di raccolta dati per l'analisi posta come obbiettivo dell'applicativo.

Il parametro *subset* restringe l'operazione alla singola colonna SPEAKER, iterando lungo le righe (*axis=0*), mentre *inplace* viene impostato a *True* per far sì che il DataFrame originale venga sostituito da quello modificato.

```
df.dropna(subset='SPEAKER', inplace=True, axis=0)
```

Figura 3.2.2: Rimozione delle entry invalide

Estraiamo adesso tutti i possibili valori per *Speaker*, *Addressee*, *Logic* e *Semantics*.
Questi ci serviranno in seguito durante la creazione dell'interfaccia dell'applicativo per dare la possibilità all'utente di personalizzare quali entry del database vengono visualizzate e prese in analisi; l'user potrà ad esempio selezionare un personaggio per filtrare il database in modo da includere solo i dialoghi che lo riguardano.
La coppia "*all : EVERYONE*" viene inserita in modo da avere un valore che permetta di visualizzare l'intero database, senza applicare alcuna restrizione.

```
#Speakers Dictionary
speakers = df['SPEAKER'].unique()
speakers_dict = {'all':'EVERYONE'}
for i in speakers:
  if not pd.isnull(i):
    spearkers_dict[i] = i
```

Figura 3.2.3: Esempio di estrazione di tutti i valori per una colonna

Affrontiamo ora il problema delle molteplici colonne per Logic, Semantics e Words. In questa implementazione scegliamo di raggruppare tutti i valori all'interno di un'unica colonna per poi eliminare quelle superflue. Trattando con colonne prive di intestazione del file originale, la soluzione più conveniente risulta quella di riferirsi ad esse attraverso il loro indice intero usando al funzione *DataFrame.iloc[row_index, column_index]*.
Iterando quindi lungo le colonne (j) e le righe (i), copiamo tutti i valori non nulli nella prima colonna, ossia quella dotata di header, per poi procedere con la rimozione dal DataFrame delle colonne senza intestazione.

```
#Logic Grouping
for j in range(5,7):
    for i in range(248):
        if not pd.isnull(df.iloc[i,5]):
            df.iloc[i,4]=str(df.iloc[i,4])+", "+str(df.iloc[i,5])
    df.drop(columns="Unnamed: "+str(j), inplace=True)
```

Figura 3.2.4: Raggruppamento dei valori delle colonne di Logic

Dal momento che non tutte le righe contengono il numero massimo di valori, è necessario eseguire un controllo tramite *pd.isnull()* per evitare di copiare all'interno della colonna finale un valore nullo e generare un errore.

3.3 User Interface

Dopo avere ottenuto un DataFrame che soddisfi le nostre esigenze, possiamo passare all'implementazione delle funzioni di Shiny per la visualizzazione della tabella e dei grafici con le informazioni sui valori di Logic e Semantics.
Dovremmo anche creare all'interno della UI un modo per l'utente di applicare dei vincoli e visualizzare i grafici di un qualunque sottoinsieme di entry.
La maggior parte della pagina sarà ovviamente occupata dal database, sotto forma di griglia.

```
#Main Panel
ui.panel_main(ui.output_data_frame("grid"))
```

Figura 3.3.1: Funzione di Shiny per la creazione di un main panel con una tabella

Per la selezione dei vincoli creiamo invece una *sidebar*, che andrà quindi ad affiancarsi alla tabella del database.
La personalizzazione avviene tramite quattro menù *drop-down*, uno per ognuno tra *Speaker*, *Addressee*, *Logic* e *Semantics*. In ognuno di questi menù sarà possibile selezionare uno dei valori presenti nel dizionario corrispondente a quella categoria, modificando dinamicamente quali dialoghi sono visibili.

```
#Sidebar menu
ui.layout_sidebar(
        ui.panel_sidebar(
            #Speaker Menu
            ui.input_select(
                "speaker",
                ui.HTML("<em><b>Speaker</b></em>"),
                speakers_dict,
                selected="all"
            ),
        [...]
```

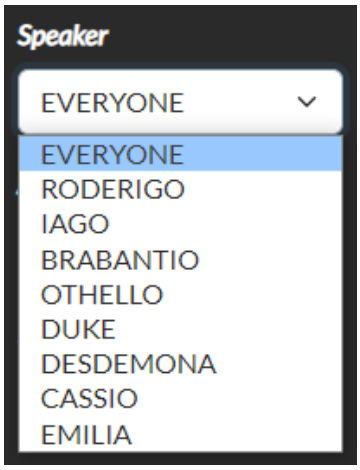


Figura 3.3.2 e 3.3.3: Codice per il menù di selezione dello Speaker e la visualizzazione nell'interfaccia

Invocando la funzione *ui.HTML()* di Shiny, possiamo anche modificare l'apparenza dell'interfaccia, in questo caso permettendoci di aggiungere i tag per il corsivo (<*em*>) e per il grassetto (<*b*>).

Per ultimo, posizioniamo i grafici con le frequenze dei valori di Logic e Semantics in cima alla pagina, in modo che siano immediatamente visibili, anche quando vengono modificati in seguito ad un cambiamento dei vincoli applicati al database.

```
#Frequency Plots
ui.column(4, ui.output_plot("logic_plot")),
ui.column(8, ui.output_plot("semantics_plot")),
```

Figura 3.3.4: Allocazione dello spazio per i grafici all'interno della pagina

Predisponiamo uno spazio maggiore per il grafico di frequenza di Semantics (8 unità su un totale di 12) al fine di aumentare la leggibilità del grafico finale, essendo i possibili valori di Semantics in quantità maggiore rispetto a quelli di Logic.

3.4 Server

Per progettare un applicativo utilizzando Shiny è necessaria anche definire l'aspetto del server, in modo che quello che viene visualizzato corrisponda alle richieste dell'utente, permettendo di modificare dinamicamente l'output qualora venissero inseriti nuovi parametri tramite l'UI.
Per ottenere un database che presenti solamente i dialoghi richiesti dall'utente, creiamo una funzione *r()* che restituisce in output la griglia modificata in base ai vari input, richiamando i valori del dizionario selezionati dall'utente tramite la *sidebar*.

```
#Reactive Grid
@reactive.Calc
    def r():
        if input.speaker() == "all": out = df
        else: out = df[df.SPEAKER==input.speaker()]

        if input.addressee() =="all": out = out
        else: out = out[out.ADDRESSEE==input.addressee()]

        if input.logic() =="all": out = out
        else: out = out[out['LOGIC'].str.contains(input.logic(), na=False)]

        if input.sem() =="all": out = out
        else: out = out[out['SEMANTICS'].str.contains(input.sem(), na=False)]

        return out
```

Figura 3.4.1: Codice per la modifica del DataFrame in base ai filri applicati

Definendo la tabella utilizzando *@reactive.Calc* facciamo sì che questa venga aggiornata qualora dovesse ricevere dei nuovi input dall'utente. Tutte le eventuali modifiche si rifletteranno poi nella visualizzazione delle entry nella tabella e nei grafici di frequenza di Logic e Semantics.
Per fare un esempio, in questa implementazione, qualora l'utente selezionasse *Logic = First* tramite il menù dell'interfaccia, la tabella verrebbe dinamicamente modificata per mostrare tutti i dialoghi in cui First è presente tra i valori di Logic.

ACT	SCENE	SPEAKER	ADDRESSEE	LOGIC	SEMANTICS	WORDS
	1.1	RODERIGO	IAGO	FIRST	MANIFEST	purse, strings, know, follow
		IAGO	RODERIGO	FIRST, FOREGONE	MANIFEST, SEEMING	eyes, proof, grounds, debitor, creditor, preferment, letter, affection, old gradation, second, heir, first
black ram		IAGO	BRABANTIO	FIRST, FOREGONE	BLACK, INVISIBLE	robbed, heart, lost, old black ram, tupping, white ewe, devil
		IAGO	BRABANTIO	FIRST, FOREGONE	MANIFEST, INVISIBLE, BLACK	God, devil, service, ruffians, Barbary horse, neigh, coursers, jennets, beast with two backs
		IAGO	RODERIGO	ECHO, FIRST	INVISIBLE	hell-pains, necessity, show, flag, sign,

Figura 3.4.2: Esempio di applicazione di un filtro

Ora che il database soddisfa i parametri impostati dall'utente, possiamo definire un metodo per elaborarlo e fornirlo in output.

```
#Grid Output
@output
@render.data_frame
def grid():
  return render.DataGrid(
    r(),
    height=None,
    width='100%',
    summary=False
  )
```

Figura 3.4.3: Funzione per il render della tabella

Il prossimo passo è creare i grafici di Logic e Semantics e passarli in output per essere visualizzati.

Tramite la funzione *value_counts()*, siamo in grado di capire quante volte ogni valore appare all'interno del database.

Applicando questo metodo sulla colonna di interesse e creando un *data.plot()*, otteniamo un istogramma le cui colonne rappresentano ognuna uno dei possibili valori che quell'attributo può assumere all'interno del DataFrame, mentre la loro altezza indica il numero di volte che tale valore compare nel database.

```
#Plot Output
@output
@render.plot(alt="Logic Graph")
def logic_plot():
  data = r().value_counts('LOGIC')
  if len(data) == 0:
    return
  return data.plot(kind='bar',color='mediumaquamarine')
```

Figura 3.4.4: Funzione per la creazione di uno dei grafici di frequenza

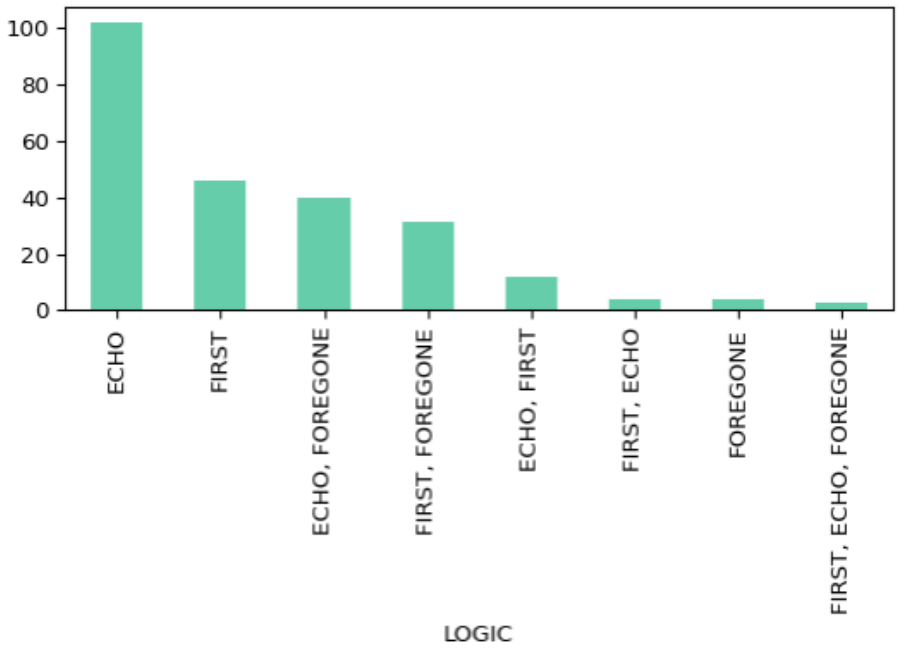


Figura 3.4.5: Grafico di frequenza per tutti i possibili valori di Logic

Per ultima cosa, non resta che unire UI e server in un unico oggetto App.

```
#UI and Server join
app = App(app_ui, server)
```

Figura 3.4.6: Creazione oggetto App

4. Risultati

Nei precedenti capitoli, abbiamo illustrato la realizzazione dell'applicativo, a partire dall'analisi del database, per poi passare alla progettazione passo per passo di un codice grazie al quale ci è possibile visualizzare e analizzare l'intero database. Il risultato è una visualizzazione tabulare del database di partenza, ma con una leggibilità migliore e con un grado di personalizzazione possibile più alto.

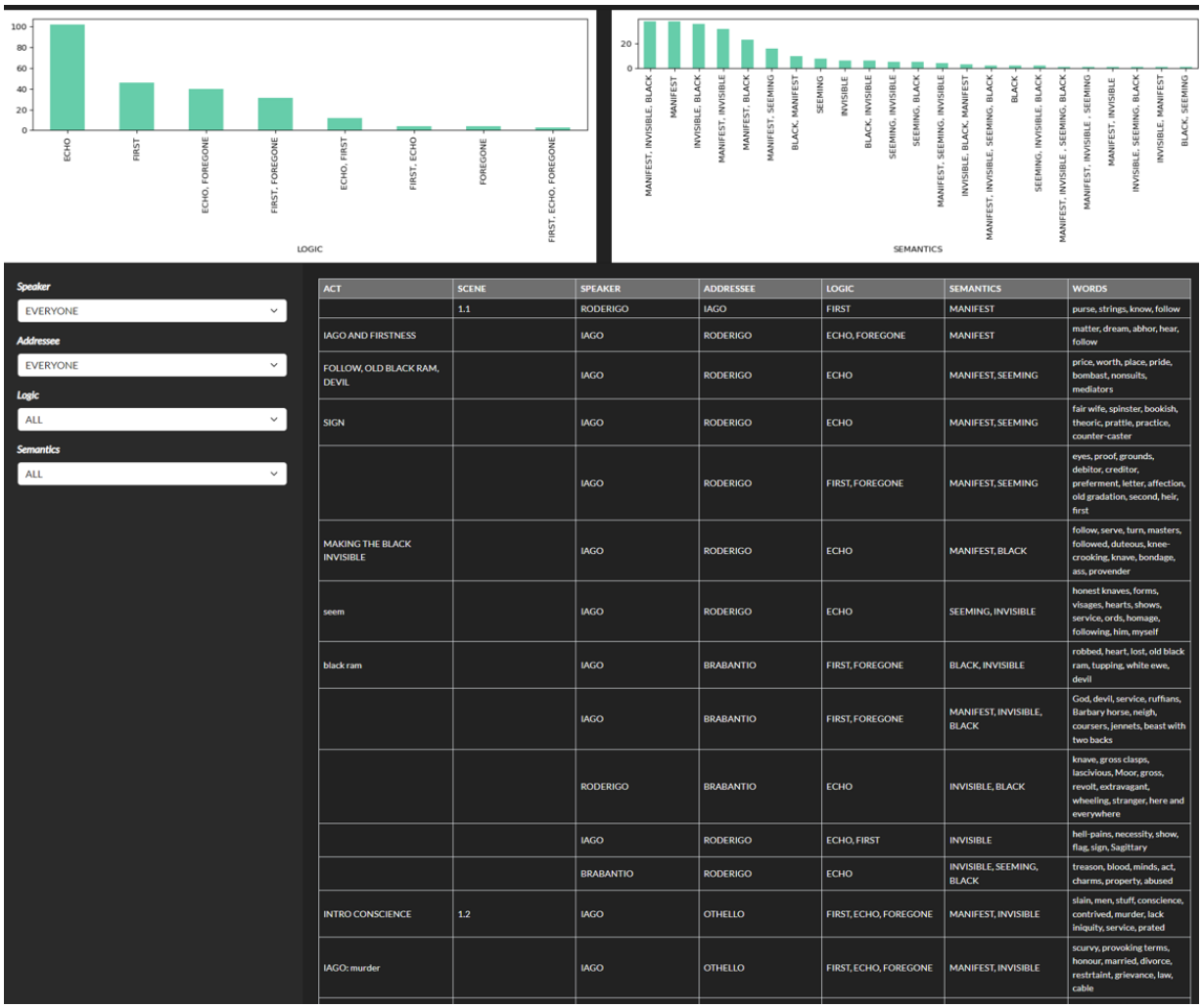
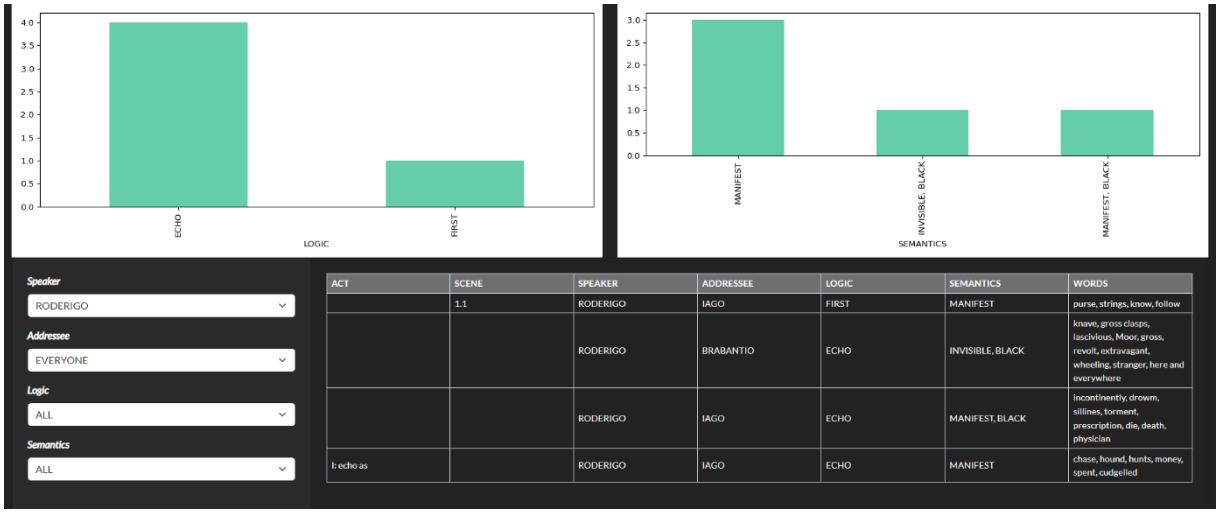


Figura 4.1: Pagina iniziale dell'applicativo

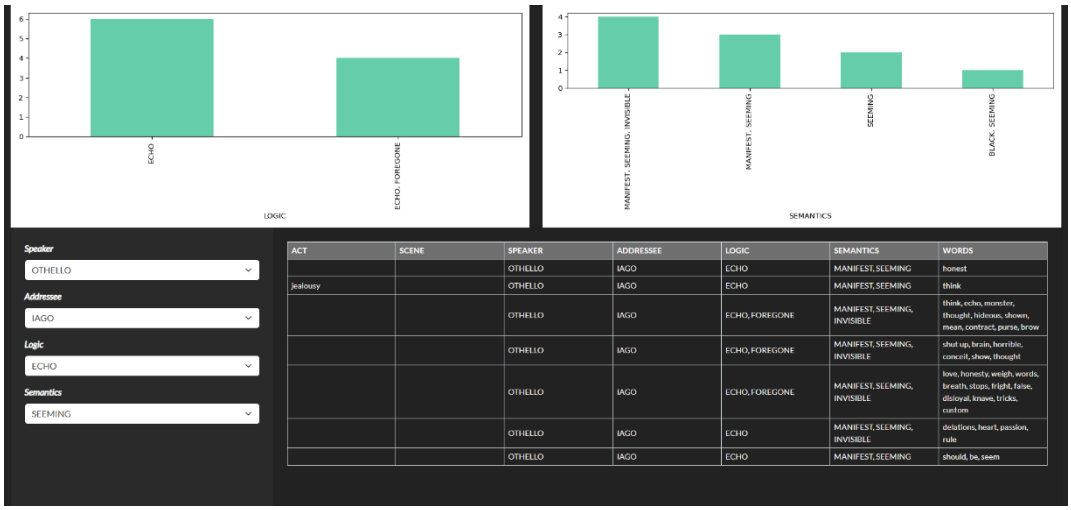
Dal grafico di Logic visto in precedenza ad esempio, possiamo osservare immediatamente come un tipo di logica *Foregone* quasi esclusivamente se accompagnata anche da *First* o *Echo*.
Vediamo anche come i vincoli modifichino sia la tabella che i grafici. Se ad esempio volessimo concentrarci sui dialoghi che vedono Roderigo come Speaker, il risultato che otterremo sarà il seguente.



ACT	SCENE	SPEAKER	ADDRESSEE	LOGIC	SEMANTICS	WORDS
	1.1	RODERIGO	IAGO	FIRST	MANIFEST	purse, strings, know, follow
		RODERIGO	BRABANTIO	ECHO	INVISIBLE, BLACK	knave, gross clasps, lascivious, Moor, gross, revolt, extravagant, wheeling, stranger, here and everywhere
		RODERIGO	IAGO	ECHO	MANIFEST, BLACK	incontinently, drowm, sillines, torment, prescription, die, death, physician
I: echo as		RODERIGO	IAGO	ECHO	MANIFEST	chase, hound, hunts, money, spent, cudgelled

Figura 4.2: Pagina dell'applicativo con applicati dei filtri

Al colpo d'occhio capiamo che il tipo di logica che predilige è *Echo* e che i suoi dialoghi sono rivolti quasi esclusivamente a Iago.
Come già detto, è possibile anche eseguire ricerche più specifiche, ad esempio applicando vincoli su tutti e quattro i campi disponibili contemporaneamente.



ACT	SCENE	SPEAKER	ADDRESSEE	LOGIC	SEMANTICS	WORDS
		OTHELLO	IAGO	ECHO	MANIFEST, SEEMING	honest
jealousy		OTHELLO	IAGO	ECHO	MANIFEST, SEEMING	think
		OTHELLO	IAGO	ECHO, FOREGONE	MANIFEST, SEEMING, INVISIBLE	think, echo, monster, thought, hideous, shown, mean, contract, purse, brow
		OTHELLO	IAGO	ECHO, FOREGONE	MANIFEST, SEEMING, INVISIBLE	shut up, brain, horrible, conceit, show, thought
		OTHELLO	IAGO	ECHO, FOREGONE	MANIFEST, SEEMING, INVISIBLE	love, honesty, weigh, words, breath, stops, fright, false, disloyal, knave, tricks, custom
		OTHELLO	IAGO	ECHO	MANIFEST, SEEMING, INVISIBLE	delations, heart, passion, rule
		OTHELLO	IAGO	ECHO	MANIFEST, SEEMING	should, be, seem

Figura 4.3: Esempio di applicazione di filtri su tutti i campi di interesse

5. Conclusioni

Nei precedenti capitoli, abbiamo illustrato il processo di progetto e sviluppo di un'applicazione Python per la visualizzazione e l'analisi di un database. Utilizzando il versatile framework *Shiny*, siamo riusciti a creare un'applicazione web interattiva che offre agli utenti la possibilità di esplorare e analizzare il database in modo dinamico.

Durante il processo di sviluppo, abbiamo eseguito un'analisi approfondita del database di partenza, composto da un file *.xlsx*, identificando le caratteristiche chiave e i problemi che avrebbero influenzato la progettazione del codice. Abbiamo affrontato le sfide legate alle colonne con valori multipli, alle celle vuote, alle voci incomplete e ai termini evidenziati, garantendo un'elaborazione dei dati accurata e la coerenza dell'interfaccia utente finale.

La progettazione dell'interfaccia utente è stata un elemento essenziale del progetto, con l'obiettivo di offrire una visualizzazione chiara e interattiva dei dati. Gli utenti possono selezionare vincoli e filtri per personalizzare la visualizzazione dei dialoghi in base a criteri come il personaggio che parla, il destinatario, la logica e la semantica.

I grafici che mostrano la frequenza di valori di logica e semantica offrono una panoramica immediata del contenuto del database.

Abbiamo utilizzato librerie Python, tra cui *Pandas* e *Matplotlib*, per la gestione dei dati e la creazione dei grafici. *Shiny* ha fornito la piattaforma per creare un'interfaccia utente dinamica e interattiva, consentendo agli utenti di esplorare il database in modo efficiente.

Il risultato finale è un'applicazione che consente di esplorare in modo dettagliato le interazioni tra i personaggi, le logiche e le semantiche presenti nell'opera "Otello". Questo progetto rappresenta quindi anche un punto di incontro tra il mondo della letteratura classica e l’analisi dei dati, offrendo un potente strumento per la comprensione e l'interpretazione di un capolavoro letterario attraverso una prospettiva innovativa.

Bibliografia

1. Repository di GitHub con il database e il codice realizzato per l'applicativo

https://github.com/Calciooo/Othello-Database-Visualizer

2. Pandas Documentation

https://pandas.pydata.org/docs/

3. Matplotlib Documentation

https://matplotlib.org/stable/index.html

4. Shiny Documentation

https://shiny.posit.co/py/api/