

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in
INGEGNERIA INFORMATICA

Tecniche Analitiche per “Open Data”

Tesi di Laurea Magistrale in
DATA MINING M

RELATORE:
Prof. Claudio Sartori

PRESENTATA DA:
ANTONINO SAVALLI

SESSIONE III
ANNO ACCADEMICO 2017/18

Indice

1.	INTRODUZIONE	1
2.	STANDARD PER OPEN DATA.....	5
2.1.	STANDARD ISO/IEC 25012	5
2.2.	BERNERS-LEE 5 STAR MODEL	7
2.3.	METADATA E PIATTAFORME OPEN DATA.....	9
3.	DATA FUSION.....	12
3.1.	LE V CARATTERISTICHE DI BIG DATA	12
3.2.	BIG DATA FUSION.....	13
3.3.	METODI DI DATA FUSION	14
4.	RECORD LINKAGE.....	16
4.1.	DATA CLEANING NEI BIG DATA	16
4.2.	EVOLUZIONE DEI METODI DI RECORD LINKAGE	17
4.3.	ALGORITMI DI COMPARAZIONE DI STRINGHE	18
5.	STRUMENTI E METODI.....	21
5.1.	LINGUAGGIO PYTHON.....	21
5.2.	LIBRERIA PANDAS	22
5.3.	LIBRERIA JELLYFISH.....	24
5.4.	LIBRERIE DI DATA CLEANING E UTILITY	25
6.	IMPLEMENTAZIONE	27
6.1.	FASE DI RICERCA	28
6.2.	FASE DI ESTRAZIONE	28
6.3.	FASE DI DATA CLEANING	30
6.4.	FASE DI TEXT MINING	31
6.5.	FASE DI RECORD LINKAGE.....	33
6.5.1.	<i>Misura di Differenza tra Stringhe nel Nome degli Attributi.....</i>	<i>33</i>
6.5.2.	<i>Record Linkage sui Valori degli Attributi</i>	<i>33</i>
6.5.3.	<i>Fase di Fusione dei Risultati.....</i>	<i>35</i>
7.	RISULTATI SPERIMENTALI.....	36
7.1.	METODOLOGIA DI TEST	36
7.2.	TEST SU BENI CONFISCATI ALLA MAFIA.....	37
7.3.	TEST SU ANAGRAFICA MUSEALE.....	43
8.	CONCLUSIONI E SVILUPPI FUTURI.....	51
	BIBLIOGRAFIA.....	55
	APPENDICE	58
A.1.	CODICE SORGENTE	58
A.1.1.	<i>Funzione ext().....</i>	<i>58</i>
A.1.2.	<i>Funzione find_claims().....</i>	<i>63</i>
A.2.	DATASET	64
A.2.1.	<i>Test su Beni Confiscati alla Mafia</i>	<i>64</i>
A.2.2.	<i>Test su Anagrafica Museale</i>	<i>70</i>
	RINGRAZIAMENTI.....	74

1. Introduzione

L'ultimo decennio ha reso estremamente popolare il concetto di Open Government [1], un modello di amministrazione aperto che fonda le sue basi sui principi di trasparenza, partecipazione e collaborazione. La sua ascesa comincia nel 2009 con l'*Open Government Initiative* [2] sotto l'amministrazione statunitense di Barack Obama, a cui seguì la fondazione il 20 Settembre 2011 dell'Open Government Partnership (OGP) [3]. L'Italia aderisce all'OGP già nell'anno della sua fondazione e presenta il Primo Piano D'azione italiano nell'aprile del 2012, impegnandosi alla diffusione dei dati aperti al fine di migliorare i propri livelli di trasparenza e integrità.

L'applicazione di questi concetti si lega a quelli degli Open Data [4], dati accessibili gratuitamente e liberamente da tutti con il fine di creare e condividere conoscenza che possa migliorare la qualità della vita del cittadino, nel caso in cui si trattino di dati governativi, come mostrato in Figura 1.

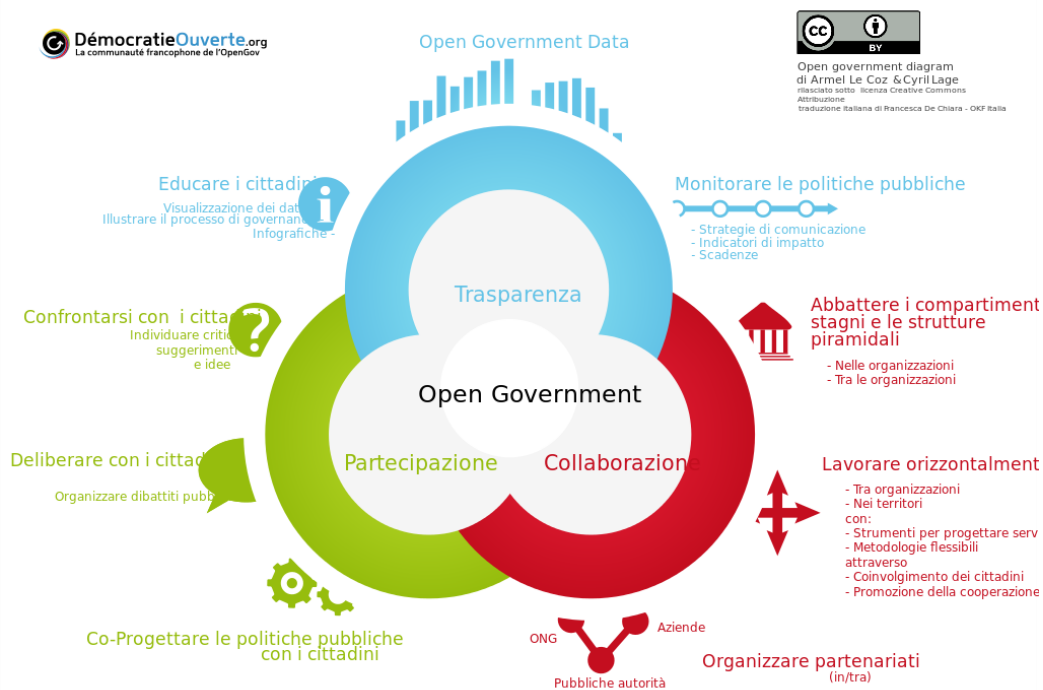


Figura 1: Open Government Data

Nel 2011, nasce il progetto Dati.gov.it, un portale che ha il ruolo di “*catalogo nazionale dei metadati relativi ai dati rilasciati in formato aperto dalle pubbliche amministrazioni italiane*” [5], con ultima versione datata Marzo 2017. Esso fornisce la possibilità di cercare e usare, anche per mezzo di API, dati aperti inseriti dagli enti pubblici italiani. *Dati.gov.it* contiene una dettagliata documentazione sugli attuali standard per gli Open Data, suggerendo quali formati e codifiche utilizzare, in particolare UTF-8, ma senza porre alcuna limitazione durante l’inserimento delle informazioni. Questa scelta ha permesso, ad oggi, il caricamento sulla piattaforma di oltre ventimila dataset, ma anche l’accumulo di un’eterogeneità di formati e informazioni molto spesso difficilmente confrontabili tra loro.

Il nostro obiettivo con il presente progetto di tesi è fornire agli utenti, tanto ai privati quanto alle imprese, un efficace strumento per ricercare, usare e confrontare le informazioni presenti sul portale *Dati.gov.it*, individuando tra i dataset similarità che possano risolvere e/o limitare l’eterogeneità dei dati presenti. In particolare, abbiamo implementato una libreria contenente funzioni che permettono la ricerca per parola chiave sul portale, il download dei dataset, l’apertura di quest’ultimi attraverso una struttura comune e tecniche analitiche per il loro confronto. Le tecniche analitiche per Open Data sono la parte centrale della tesi perché riteniamo che le informazioni contenute su *Dati.gov.it* siano di estrema importanza. Nonostante attualmente può essere molto laborioso estrarle, gli strumenti e metodi di Data Mining utilizzati in questo progetto potrebbero fornire l’atteso valore alle informazioni presenti.

Abbiamo diviso il progetto in tre parti fondamentali equamente divise in fase di studio e di sviluppo. Durante la prima fase è stato individuato in Python [6] il linguaggio di programmazione da utilizzare, poiché estremamente adatto allo sviluppo di moduli open source grazie alla sua flessibilità e nondimeno perché particolarmente usato nel contesto della Data Science grazie alla presenza di utili librerie come *Pandas* [7], largamente utilizzato per questo progetto tramite la struttura *Data Frame* e *Jellyfish* [8], che fornisce i più importanti algoritmi di misurazione di stringhe. Inoltre, in questa fase, sono stati individuati gli standard uso per Open Data, Metadata e piattaforme di Open Data in uso, sono state analizzate le Web API del portale *Dati.gov.it* ed è stata implementata la funzione di

ricerca per parole chiave all'interno del portale. Durante la seconda fase, ci siamo concentrati sull'estrazione dei dataset dal portale tramite l'utilizzo di una struttura comune, che nel nostro caso è *Data Frame* della libreria *Pandas*.

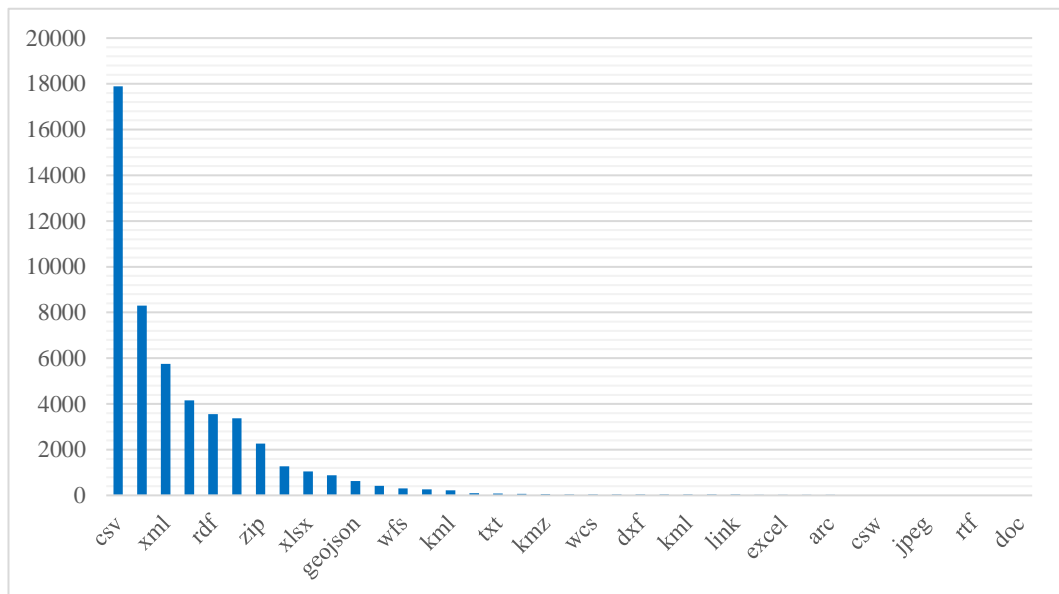


Figura 2: occorrenza dei formati sul portale Dati.gov.it

L'implementazione ha tenuto conto di tutti i problemi causati dall'eterogeneità dei dati con particolare cura nell'estrarre i formati più presenti. Infatti, come mostrato in Figura 2, la piattaforma contiene, rispetto al totale, più del 78% di dataset in formato *csv*, il 36% in *json*, il 25% in *xml*, il 18% in *xls*, il 15% in *rdf*, il 14% in *html* e altri quaranta formati con un'occorrenza inferiore al 10%. Per questo motivo, i formati supportati sono, in ordine di priorità e quindi di *feature* sviluppate: *csv*, *json*, *xml*, *xls*, *rdf*, *pdf* e *txt*. Considerando l'alta presenza di file in formato *csv*, è stata posta particolare attenzione alle sua gestione e controllo degli errori. D'altro canto, la parte di studio per la fase successiva si è focalizzata sul confrontare e omogeneizzare i dati sfruttando metodi e algoritmi di *Data Fusion* e di *Record Linkage*.

Infine, durante la terza fase, abbiamo usato metodi e algoritmi utilizzati nel campo del *Data Fusion* e del *Record Linkage* al fine di fornire delle funzioni che permettessero, dopo una fase di *Data Cleaning*, la ricerca di similarità tra stringhe di due dataset. Le due funzioni principali forniscono metodi per trovare

un'omogeneità tra i nomi degli attributi di due dataset e tra i valori degli attributi per mezzo dello sviluppo di un algoritmo che verifica l'importanza delle parole all'interno dei dataset.

La dissertazione è stata divisa in otto capitoli, di cui in questo primo capitolo abbiamo appena affrontato l'introduzione.

Nei capitoli due, tre e quattro analizzeremo lo stato dell'arte del contesto preso in considerazione. In particolare, nel capitolo due affronteremo i temi riguardanti gli standard degli Open Data con focus sulla Data Quality, gli standard principali dei Metadata e gli standard sulle piattaforme di metadata. Nel capitolo tre, introdurremo il contesto dei Big Data e analizzeremo i metodi di Data Fusion. Nel capitolo quattro, termineremo la valutazione del contest di studi analizzando l'importanza e la difficoltà del *Data Cleaning* nei Big Data, i metodi forniti dal Record Linkage con particolare attenzione per gli algoritmi di comparazione tra stringhe.

Nel capitolo cinque forniremo una panoramica degli strumenti e metodi utilizzati durante la tesi che hanno permesso l'implementazione mostrata in dettaglio nel capitolo sei.

A conferma del lavoro svolto, nel capitolo sette, mostreremo i risultati ottenuti con due esperimenti. Il primo contiene due dataset sui beni confiscati alla mafia, uno per la città di Milano e uno per la città di Catania. Il secondo contiene due dataset con molte informazioni sui musei della regione Lazio in uno e della regione Sicilia nell'altro.

Infine, nel capitolo otto, termineremo la dissertazione con le conclusioni e i possibili sviluppi futuri.

2. Standard per Open Data

Un problema già noto nel campo dei Big Data è la necessità di ricercare e collegare informazioni seguendo una linea semantica. Le possibili soluzioni sono state incluse negli standard che analizziamo in questo capitolo: gli standard in utilizzo per gli Open Data, per i Metadata e per le piattaforme di Big Open Data.

Nel paragrafo 2.1 e 2.2 trattiamo gli standard attuali di *Big Data Quality* con particolare attenzione per il modello ISO standard 25012 nel paragrafo 2.1 e il *Berners-Lee 5 star model* nel paragrafo 2.2. Nel paragrafo 2.3, ci concentriamo sugli standard dei metadata e sulle principali piattaforme di Open Data con particolare interesse per *CKAN*, la piattaforma usata dal nostro portale Open Data di riferimento Dati.gov.it.

2.1. Standard ISO/IEC 25012

L'organizzazione internazionale per la normazione (ISO) ha definito nel 2008 lo standard 25012 dal titolo: "*Data Quality Model*" [9], divenuto legge nel 2014 con la sigla UNI CEI ISO/IEC 25012. Lo standard ISO 25012 ha lo scopo di:

- a. definire e valutare i requisiti nella produzione, acquisizione e integrazione dei dati;
- b. identificare i criteri di qualità dei dati;
- c. valutare la conformità dei dati rispetto alle leggi nazionali e/o requisiti già esistenti.

Come mostrato in Figura 3, il modello di qualità dei dati è composto da 15 caratteristiche classificate in due categorie principali, *inerenti e dipendenti dal sistema*, e la loro intersezione.

Le qualità dei *dati inerenti* si riferiscono alle caratteristiche che hanno il potenziale intrinseco per soddisfare i bisogni dichiarati e impliciti. In particolare, si prendono in considerazione i valori del dominio dei dati, le relazioni tra i valori dei

dati e i metadata. Le caratteristiche inerenti definite sono: accuratezza, attualità, coerenza, completezza e credibilità.

Le qualità dei *dati dipendenti dal sistema* si riferiscono alle caratteristiche presenti all'interno di un sistema informatico, ossia quando i dati sono usati in specifiche condizioni. Appartengono a questa categoria le caratteristiche di disponibilità, portabilità e ripristino.

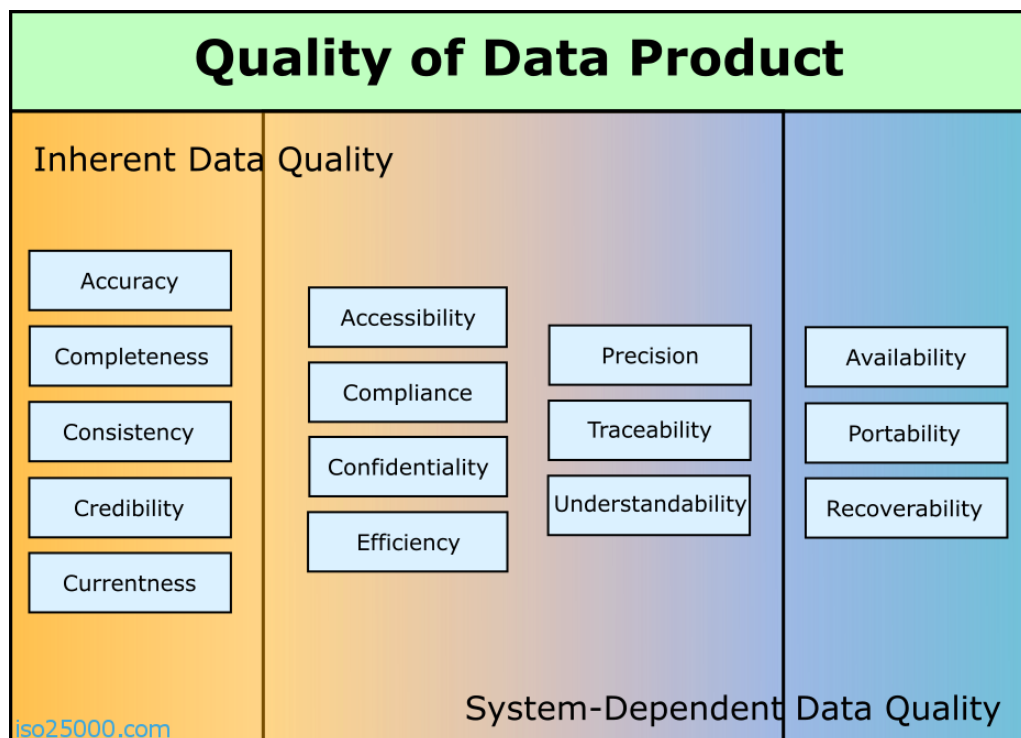


Figura 3: caratteristiche divisi per categorie nel modello ISO 25012

Infine, le qualità sia *inerenti* che *dipendenti dal sistema* si riferiscono alle caratteristiche che abbracciano entrambe le categorie principali. Appartengono a questa categoria le caratteristiche di accessibilità, comprensibilità, conformità, efficienza, precisione, riservatezza e tracciabilità.




2.2. Berners-Lee 5 star model

Tim Berners-Lee, co-inventore del World Wide Web, scrisse nel 2006 un importante articolo sui *Linked Data* [10], in cui spiegava come non sia più sufficiente inserire materiale su internet, ma bisogna collegare tutte le informazioni tra loro. In particolare, l'autore trova soluzione al problema nel formato RDF, *Resource Description Framework* [11], basato su tre principi fondamentali:

- a. Qualsiasi cosa può essere espressa tramite *Uniform Resource Identifier* (URI);
- b. utilizzare il linguaggio meno espressivo possibile per definire qualcosa;
- c. Qualunque cosa può dire tutto su qualsiasi cosa.

Nello specifico, il formato RDF, serializzabile anche nel più noto formato XML, è rappresentabile tramite grafo orientato permettendo una connessione potenzialmente infinita tra le informazioni presenti su internet.

Il *Berners-Lee 5 star model* per gli Open Data nasce come un aggiornamento, datato 2010, all'articolo descritto. Nell'approfondimento l'autore spiega come gli Open Data siano un ottimo campo di applicazione per i Linked Data. Nasce dunque il concetto di Linked Open Data (LOD), una classifica di Dati Aperti, da una stella fino a cinque, definiti secondo una serie di costi-benefici.

	Available on the web (whatever format) <i>but with an open licence, to be Open Data</i>
	Available as machine-readable structured data (e.g. excel instead of image scan of a table)
	as (2) plus non-proprietary format (e.g. CSV instead of excel)

★★★★	All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
★★★★★	All the above, plus: Link your data to other people's data to provide context

Tabella 1: modello 5 Star Open Data

Come mostrato in Tabella 1, gli Open Data sono catalogati in stelle che variano da una fino a cinque [2]. Nello specifico:

- a. I Dati Aperti con una stella sono disponibili sul web con una licenza aperta senza alcun vincolo sul formato scelto;
- b. due stelle sono attribuite ai Dati Aperti forniti con dei formati che possono permettere una manipolazione su larga scala. Solitamente fanno parte di questa categoria i dati con un formato che permette la manipolazioni dei dati solo tramite software proprietario. Un classico esempio è *Microsoft Excel*;
- c. tre stelle sono attribuite ai formati che permettono una modifica con software non-proprietari come CSV;
- d. quattro stelle sono attribuite ai dati forniti con formati che siano *Open Standards* per il World Wide Web Consortium (W3C) [12]. Nello specifico, ci riferiamo al formato RDF e SPARQL [13];
- e. cinque stelle sono attribuite agli Open Data con un formato da quattro stelle, ma che contengono degli accorgimenti architetturali che migliorano l'organizzazione, il valore e la comprensione del contesto dei dati forniti con collegamenti ad altri Dati Aperti.

2.3. Metadata e Piattaforme Open Data

La necessità degli Open Data di essere usufruibili e condivisibili da tutti, si confronta spesso con la realtà dei fatti, dove gli utenti non riescono a trovare le informazioni che cercano perché riscontrano frequentemente difficoltà nello scovare i dataset a cui si è interessati o non si può comprendere il suo contenuto prima di analizzarlo. Per questo motivo i metadata hanno un ruolo fondamentale all'interno del contesto dei Dati Aperti.

I metadata [14] sono delle informazioni strutturate che rendono molto più semplice trovare, usare e gestire le risorse di informazioni perché descrivono un dataset e aiutano a capirne il contenuto prima di analizzarlo.

Nel contesto degli Open Data, i metadata sono contenuti in apposite strutture denominate *piattaforme di Open Data* [15], di cui bisogna premettere che nascono per fornire informazioni su un determinato contesto.

Open data Platform	Metadata Model
CKAN	CKAN
OpenDataSoft	DCAT
SOCRATA	Socrata
DKAN	DCAT, INSPIRE
ArcGis Open Data	INSPIRE
Esri Geoportal Server	Open Geospatial Consortium (OGC) compliant CS-W 2.0.2 service
Junar	DCAT, INSPIRE

Tabella 2: i modelli di metadata usati dalle principali piattaforme di Open Data

Di conseguenza, è il contenuto delle informazioni che spinge un portale a usare una determinata piattaforma di Open Data. Tuttavia, negli anni alcune piattaforme sono state utilizzate molto più di altre per determinati campi, diventando degli *standard de facto*, anche grazie alla loro caratteristica di essere basati su almeno uno dei principali *Core Metadata Standard*, come si può constatare dalla Tabella 2.

I principali *Core Metadata Standard* [15] su cui si basano le piattaforme di Open Data più usate sono: *RDF Data Cube Vocabulary*, *Dublin Core*, *Data Catalog Vocabulary* (DCAT) e *INSPIRE Metadata Schema*. *RDF Data Cube Vocabulary* è

un vocabolario che fornisce il mezzo per pubblicare dati multidimensionali sul web con la possibilità di essere collegati ad altri dataset usando lo standard RDF. Dublin Core, pubblicato come ISO Standard 15836 nel 2009, è uno standard *domain-agnostic* che può essere facilmente compreso e implementato. DCAT ha lo scopo di migliorare l'interoperabilità tra i dati, così che le applicazioni possano facilmente usare metadata da più cataloghi. DCAT è lo standard di metadata più usato grazie al suo design e alla sua flessibilità.

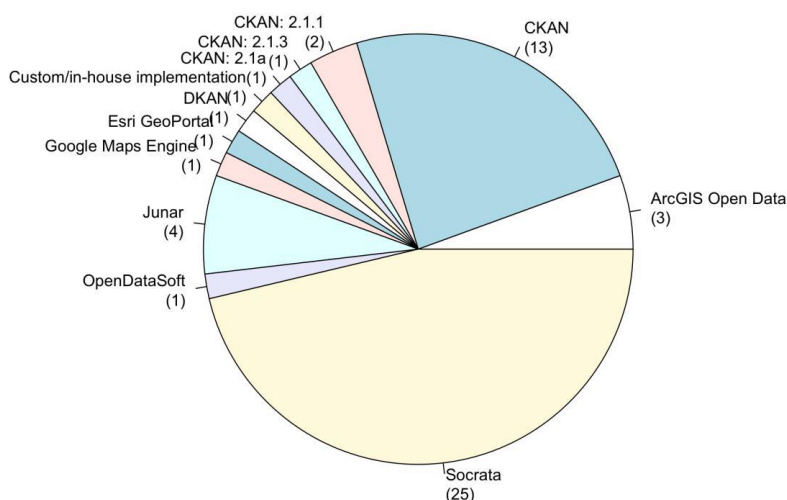


Figura 4: distribuzione delle piattaforme di metadata su un campione di 55 portali

Le piattaforme di Open Data principali sono: *CKAN*, *OpenDataSoft*, *SOCRATA*, *DKAN*, *ArcGisOpenData*, *Esri Geoportal Server*, *Junar* e come si può osservare dalla Figura 4, tratta da uno studio dell'Ottobre 2016 effettuato su un campione di cinquantacinque portali di Open data [15], le più utilizzate sono CKAN e Socrata.

Socrata è basato su metadata RDF con la possibilità di arricchirne i campi. CKAN conserva i dataset come una cartella che ospita i dataset o le risorse. In CKAN i metadata sono forniti come RDF e la piattaforma supporta i formati DCAT, Dublin Core e INSPIRE fornendo diversi campi che descrivono il dataset con la possibilità di aggiungere dei campi extra personalizzabili, come evidenziato dalla Tabella 3.

Fields	Description
Title	Field used to label datasets. This attribute is intended to allow search, sharing and linking of datasets
Unique identifier	This attribute assigns a unique URL to a dataset. This is one of the Dublin Core recommendations
Groups	A customisable group that the dataset belongs to
Description	Human readable description of the dataset
Data preview	Quick preview in the comma separated value (CSV) format of the dataset
Revision history	Provides revision history
Licence	Allows user to check what licence a given dataset is
Tags	Allocating tags to datasets makes them more discoverable through tag search and faceting by tags
Formats	Provides information on the format datasets is available for download in
API key	Allows for a developer access to the metadata fields
Customizable extra fields	Such as location data or extra information relevant to the publisher or the dataset

Tabella 3: la struttura di CKAN

3. Data Fusion

In questo capitolo analizziamo lo stato dell'arte del *Data Fusion*, un settore di studi nato dalla necessità di diminuire i tempi di Data Cleaning in un campo fortemente eterogeneo come quello dei Big Data. Esamineremo prima, nel paragrafo 3.1, il contesto in cui ci troviamo soffermandoci sulla definizione di Big Data e sulle 5 V che lo caratterizzano. Nel paragrafo 3.2, definiremo l'area di studi del Data Fusion e i motivi per cui la comunità scientifica ne è interessata. Infine, nel paragrafo 3.3, analizzeremo gli strumenti e i metodi di Data Fusion più utilizzati.

3.1. Le V Caratteristiche di Big Data

Attualmente esistono tante e diverse definizioni di Big Data. La confusione nasce spesso dal punto di vista o settore di interesse dell'autore. Di conseguenza, nonostante le diverse definizioni di Big Data siano corrette, l'eterogeneità che ne è derivata può confondere facilmente il concetto. Per questo motivo, si è soliti definire i Big Data prendendo in considerazione la sua caratterizzazione con l'utilizzo delle 3 V, coniate da Doug Laney [16] a cui se ne aggiunge spesso almeno un'altra.

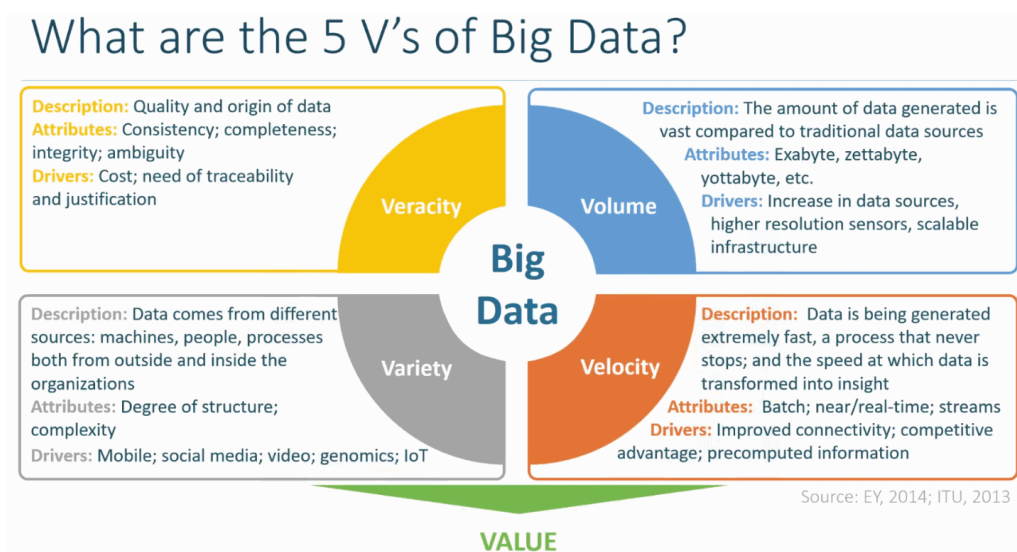


Figura 5: Le 5 V di Big Data

Le 4 V dei Big Data sono: volume, velocità, varietà e veracità. Nello specifico, il Volume indica la grande quantità di dati generati nel breve lasso di tempo; la Velocità indica la rapidità con cui i dati si diffondono; la Varietà indica le diverse tipologie di dati che sono contenuti nel termine Big Data come testo, audio, video, dati geo spaziali; infine, Veracità indica la qualità dei dati di ogni fonte, che può essere fortemente diversa.

Solitamente, come si evince dalla Figura 5, è presente una quinta V che fonde le caratteristiche precedenti: il Valore. Esso indica che i dati forniti dai sistemi di Big Data devono essere accurati e fornire un misurabile miglioramento delle informazioni a disposizione.

3.2. Big Data Fusion

La Data Fusion ha lo scopo di garantire la qualità e l'estrazione analitica di dati integrati. La sua utilità si nota citando uno studio che spiega come almeno il 70% del tempo speso nell'analisi dei dati non è speso nell'analisi in sé, bensì nel processo di trovare, interpretare, estrarre e ricombinare i dati analizzati. Di conseguenza, la Data Fusion nasce con l'ottica di diminuire drasticamente i tempi di Data Cleaning fornendo dei metodi che possano, non solo ridurre i tempi precedenti all'analisi, ma anche aumentare il valore dei Big Data su cui si lavora.

La Fusione dei Dati è divisa in tre livelli [17]:

- a. il Data Layer Fusion è un'integrazione di basso livello, in cui si aggiungono dei dati sui data originali e si analizzano;
- b. il Feature Layer Fusion prevede una fusione di livello intermedio, in cui si effettua un'estrazione delle feature dai dati originali per essere successivamente analizzati e processati sinteticamente. Questa scelta facilita i processi real-time;
- c. il Decision Layer Fusion prevede l'estrazione, l'analisi e la sintesi di tutti i dati rispettando le diverse fonti di provenienza. Successivamente, si fondono i dati tra loro per mezzo delle singole sintesi ottenute, che

devono rispettare dei termini di omogeneità per essere confrontabili tra loro.

3.3. Metodi di Data Fusion

I metodi di Data Fusion sono divisi in tre macro-categorie [17]:

- a. il metodo di Data Fusion basato su fasi consiste nella fusione di dati nel processo di analisi di Data Mining. In questo caso, i dati eterogenei si possono fondere durante l'intero processo sfruttando diverse fasi per raggiungere l'obiettivo di creare dati eterogenei multi-sorgente. Tuttavia, nel metodo basato su fasi non ci sono interazione tra i dati eterogenei, con lo svantaggio di non riuscire a superare il divario semantico e di conseguenza non raggiungere una vera fusione di dati intrinseca;
- b. il metodo di fusione di dati basato a livello di feature è un altro metodo di fusione di dati eterogenei multi-sorgente. Eseguito nel livello intermedio, sono inizialmente estratte le feature dalle fonti eterogenee. Successivamente sono analizzate e processate per formare una matrice o un vettore comune di dati eterogenei multi-sorgente. Solitamente, una volta formato un vettore di feature, esso è utilizzato per tecniche di Data Mining come la classificazione o il clustering;
- c. i metodi di fusione di dati basati sulla semantica, si dividono in quattro categorie: metodi basati sulla *multi-view*, metodi basati sulla similarità, metodi basati sulla dipendenza probabilistica e metodi basati sul *transfer learning*.

Nel campo applicativo, le varie categorie spesso si incrociano in un unico metodo. In particolare, i principali algoritmi di Big Data Fusion fanno parte di tre macro-aree [18]:

- a. Il *riconoscimento delle named entity* ha il compito di identificare specifiche categorie di entità a partire dai testi. Gli studi principali in materia risalgono al 2002, quando Chua et al. riescono a trovare regole tra sinonimi tra i nomi cinesi e le strutture organizzati con l'utilizzo di *HowNet*. Successivamente, nel 2007, Mihalcea usa Wikipedia come dizionario per identificare entità di interesse. Attualmente gli studi ci conducono a un utilizzo della *semantic Web theory* insieme ai più noti modelli di machine learning. Lo scopo è unire i grandi potenziali di ricerca di entità del Web con la capacità delle tecniche di machine learning al fine di trovare pattern rilevanti;
- b. *l'unificazione identificata delle entità* è il processo che unisce differenti rappresentazioni di un'unica entità del mondo reale. Particolarmente utilizzata nel mondo delle immagini, l'idea di base è di ordinare vari attributi dell'entità e poi fondere tutti i risultati ordinati per ottenere i migliori accoppiamenti. L'utilizzo di funzione di kernel di sequenze semantiche ha permesso di usare con successo questo metodo anche nel mondo delle parole;
- c. lo *studio della risoluzione del conflitto dei dati* è la naturale conseguenza dei processi su dati eterogenei. Infatti, nel campo della Big Data Fusion è frequente avere conflitti di classificazione. I metodi per la loro risoluzione sono divisi in due categorie: i *metodi sul rapporto basato sull'espansione* e i *metodi basati sulla selezione di valori veri provenienti da valori in conflitto multiplo*.

Proposto da Bleiholder nel 2008, il primo metodo usa soprattutto operazioni relazionali estese e funzioni di aggregazione. Il metodo richiede esperti del dominio per specificare le differenti funzioni di risoluzione dei conflitti. Il secondo metodo proposto da Wu et al. utilizza un criterio di selezione di *valori veri* a cui si aggiungono i risultati di un motore di ricerca basato sull'importanza e la similarità tra sorgenti di dati Web.

4. Record Linkage

In questo capitolo analizziamo l'importanza del Record Linkage nel contesto del Data Cleaning e dell'*Entity Resolution*, indicando i principali metodi e algoritmi utilizzati per trovare collegamenti all'interno di fonti eterogenee della stessa entità. Nel paragrafo 4.1 spieghiamo l'importanza della fase di Data Cleaning nei Big Data, campo in cui il record linkage è estremamente utile. Nel paragrafo 4.2, forniamo un *excursus* che permette di comprendere l'evoluzione del record linkage e ci concentriamo sui metodi più usati con particolare interesse per gli *algoritmi di misurazione di similarità tra stringhe* che sono approfonditi nel paragrafo 4.3.

4.1. Data Cleaning nei Big Data

Il termine Record Linkage è fondamentale nei Big Data perché strettamente associato a quello del *Data Cleaning* o *pre-process*. Considerando che l'ottanta per cento del tempo di un *data scientist* è dedicato alla pulizia di dati [19], trovare nuovi metodi e algoritmi per velocizzare o addirittura rendere possibile la pulizia dei dati risulta di vitale importanza.

I dataset contengono spesso campi duplicati, record che si riferiscono alla stessa entità nel mondo reale o semplici errori di battitura che rendono due valori non identici. Il Record Linkage si prefigge l'obiettivo di eliminare o almeno ridurre l'ambiguità nelle rappresentazioni di istanze con lo scopo di individuare quali elementi simili rappresentino lo stesso elemento e quali no [20]. In un contesto eterogeneo come quello dei Big Open Data, il Record Linkage può essere molto complicato. Inizialmente bisogna ripulire i dati in modo che errori tipografici o particolari scelte dell'utente possano prevenire il riconoscimento di entità simili. In seguito, bisogna selezionare i campi che saranno utilizzati nel processo di similarità tra i record. Infine, bisogna decidere quale sia l'algoritmo di similarità più adatto in base alle esigenze e scegliere il corretto valore di soglia che permetta di intercettare soltanto i record simili che hanno in comune la stessa entità nel mondo reale. Considerando che i Big Open Data sono strutture molto eterogenee, voluminose e

in continua evoluzione, si può già intuire quanto possa essere complicato e oneroso [20].

4.2. Evoluzione dei Metodi di Record Linkage

Record Linkage è un termine che risale al 1946, titolo di un articolo di Halbert L. Dunn [21]. Successivamente, il termine fu usato nuovamente per il lavoro pionieristico di Ivan Fellegi e Alan Sunter dal titolo “*A Theory For Record Linkage*” contenente il formale modello matematico noto come “*Modello di Record Linkage di Fellegi-Sunter*” [22]. Successivamente, in particolare dagli studi di Matthew A. Jaro del 1989 [23], il Data Cleaning si concentra sulla comparazione di stringhe e nella standardizzazione di nomi e indirizzi anche grazie ai metodi e agli strumenti sviluppati da William E. Winkler a partire dal 1990 [24].

Il metodo usato nel modello di Record Linkage di Fellegi-Sunter [22] si basa su un approccio probabilistico per risolvere il problema del record linkage su un modello decisionale. I record contengono attributi che identificano una singola entità che sono utilizzati per testare il metodo.

GENERAL RECORD LINKAGE SYSTEM

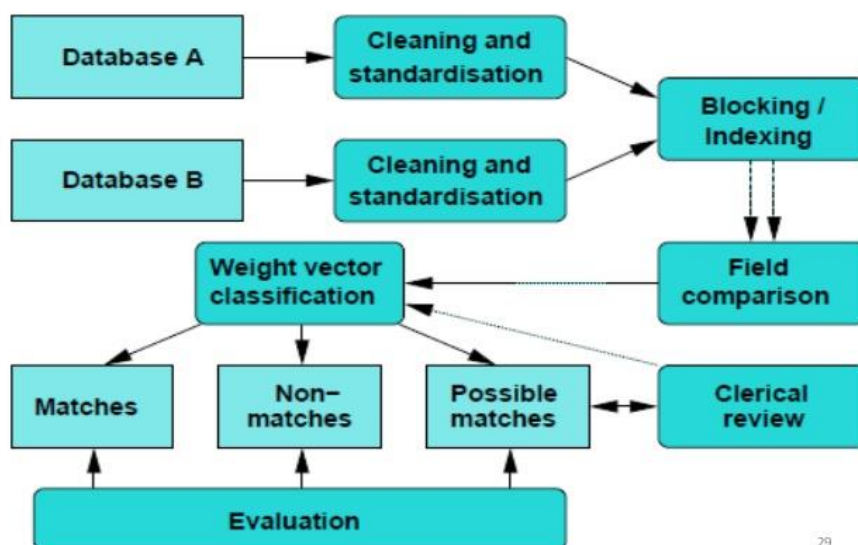


Figura 6: sistema generale di Record Linkage

Come si nota dalla Figura 6, esso prevede che date due sorgenti, tutte le coppie provenienti dal prodotto cartesiano devono essere classificate in tre subset indipendenti e mutuamente esclusivi: il set dei match, il set dei non-match, e il set con coppie che richiedono un controllo manuale. Per classificare le coppie, gli attributi in comune sono usati per stimare la probabilità di ogni coppia di appartenere sia al set dei match che dei non-match. Il criterio di classificazione delle coppia è basato sul rapporto tra condizioni di probabilità. Il modello decisionale ha lo scopo di minimizzare sia gli errori di *misclassification* che la probabilità di una coppia di appartenere alla classe di revisione manuale.

La standardizzazione di nomi e indirizzi è composta da due fasi [24]. La prima è la standardizzazione vera e propria, che consiste nel sostituire lo *spelling* di diverse parole in un'unica con l'ausilio di tabelle di *lookup* contenenti lo spelling standard di molte parole. Successivamente, nella seconda fase denominata *parsing*, le parole sono divise in componenti che possono essere confrontati.

La comparazione di stringhe nasce dalla reale e frequente evenienza in cui non è possibile confrontare due stringhe a causa di errori tipografici. Il Record Linkage risponde a questo problema con algoritmi di comparazione di stringhe che permettono di comprendere quanto simili siano tra loro due stringhe [24]. La distanza tra stringhe può essere calcolata in diversi modi, ma la loro applicazione è strettamente connessa all'utilizzo di valori di soglia che richiedono all'utente un'accurata analisi e conoscenza del contesto di applicazione. In ordine cronologico di introduzione, gli algoritmi di comparazione di stringhe più noti sono: la Distanza di Hamming, la Distanza di Levenshtein, la Distanza Damerau-Levenshtein, la *Match Rating Approach Comparison*, la Distanza di Jaro e la Distanza di Jaro-Winkler.

4.3. Algoritmi di Comparazione di Stringhe

La Distanza di Hamming tra due stringhe di uguale lunghezza è il numero di posizioni nelle quali i simboli corrispondenti sono diversi [25]. Di conseguenza, indica il numero di sostituzioni necessarie ottenere l'eguaglianza tra le due stringhe.

L'algoritmo di Hamming non ha una forte applicazione nel mondo del Record Linkage perché impone il vincolo di stringhe di uguale lunghezza, ma è alla base di altri algoritmi più sofisticati come la Distanza di Levenshtein.

La Distanza di Levenshtein calcola la similarità tra due stringhe fornendo il numero minimo di modifiche elementari necessari per ottenere un'eguaglianza tra le stringhe. Le modifiche possono essere: la cancellazione di un carattere, la sostituzione di un carattere con un altro o l'inserimento di un carattere. Di conseguenza, è possibile calcolare la Distanza di Levenshtein anche per stringhe con lunghezza differente tra loro.

La Distanza di Damerau-Levenshtein differisce dalla Distanza di Levenshtein per l'inclusione dell'operazione di trasposizione in aggiunta alle tre modifiche elementari su singolo carattere [26]. È particolarmente usato nel campo del *Natural Language Processing*, nel confronto tra filoni di DNA e nel capo della detenzione delle frodi.

Nel 1977 fu sviluppato l'algoritmo fonetico di *Match Rating Approach Comparison* per l'indicizzazione e il confronto di nomi omofoni. Si compone di due fasi: una di codifica e una di confronto [27]. Nella fase di codifica, sono eliminate tutte le vocali tranne quelle presenti a inizio della stringa e tutte le consonanti di ogni doppia consonante presente. Il risultato finale è un codice di sei lettere con le prime e ultime tre lettere dell'input modificato. Nella fase di confronto, se la differenza tra i sei valori è maggiore o uguale a tre, il risultato è di dissimilarità. In caso contrario, è effettuata un'analisi di similarità analoga ai già esistenti algoritmi di confronto tra stringhe come la Distanza di Hamming o la Distanza di Levenshtein con l'aggiunta dell'utilizzo di un valore minimo di soglia.

La Distanza di Jaro, introdotta nel 1989, consiste nell'utilizzo delle operazioni di inserimento, eliminazione e trasposizione [23]. L'algoritmo si divide in tre fasi. La prima fase consiste nel computare la lunghezza delle stringhe. La seconda fase consiste nel trovare il numero comune di caratteri tra le stringhe. La terza fase consiste nel trovare il numero di trasposizioni. La definizione di *carattere comune* è verificata quando il carattere concordante è entro la metà della lunghezza della stringa più piccola. La definizione di *trasposizione* è verificata quando un carattere

di una stringa è fuori posizione rispetto un *carattere comune* dell'altra stringa. Il risultato di confronto tra stringhe, normalizzato tra zero e uno, è dato da:

$$\Phi_j(s_1, s_2) = \frac{1}{3} \left(\frac{N_c}{len_{s_1}} + \frac{N_c}{len_{s_2}} + 0.5 \frac{N_t}{N_c} \right)$$

con s_1 e s_2 corrispondenti alle stringhe confrontate, con relativa lunghezza len_{s_1} e len_{s_2} . N_c è il numero di caratteri in comune tra le stringhe s_1 e s_2 . N_t è il numero di trasposizioni.

L'anno successivo alla formulazione dell'algoritmo di Jaro, Winkler pubblica un miglioramento che darà vita all'algoritmo di Distanza di Jaro-Winkler [28]. Esso aggiunge un prefisso di scala p che fornisce una votazione più alta alle stringhe che hanno in comune un prefisso di lunghezza l all'inizio della stringa. La similarità di Jaro-Winkler è definita come:

$$sim_w = sim_j + lp(1 - sim_j)$$

con sim_j la similarità di Jaro tra le stringhe. La costante l può avere valore massimo di quattro, mentre la costante p ha come standard valore 0.1 e non dovrebbe superare 0.25 per evitare che la Distanza valga più di uno. La Distanza di Jaro-Winkler è definita come:

$$d_w = 1 - sim_w$$

5. Strumenti e Metodi

Mostriamo ora gli strumenti e i metodi utilizzati durante l'implementazione della tesi e i motivi per cui sono stati scelti. Nel paragrafo 5.1, mostriamo l'utilità del linguaggio di programmazione Python e i motivi che ci hanno spinto ad usarlo. Lo stesso faremo nei successivi paragrafi con particolare attenzione per la libreria Pandas nel paragrafo 5.2, per la libreria Jellyfish nel paragrafo 5.3 e per le librerie di Data Cleaning e Utility nel paragrafo 5.4.

5.1. Linguaggio Python

In una ricerca su un campione di sedicimila lavoratori nel campo dei dati è stato chiesto, tra le varie domande, quale fosse il linguaggio di programmazione che usassero. Python risulta essere lo strumento più usato nel campo dei dati [29], come mostrato in Figura 7.

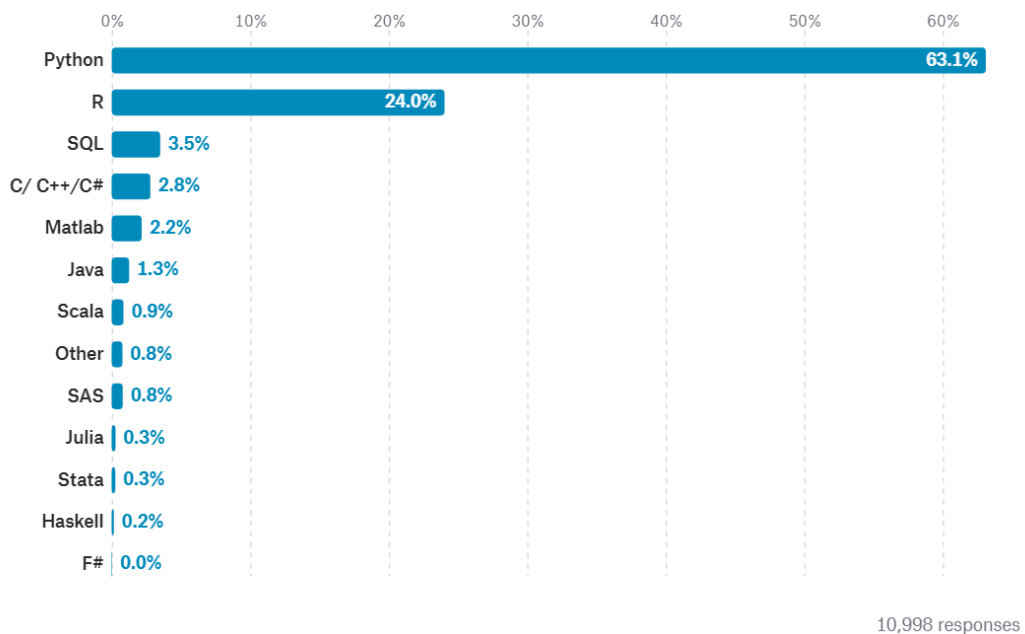


Figura 7: il 63% degli utenti che hanno partecipato alla ricerca di kaggle.com usa Python

Python è un linguaggio di alto livello orientato agli oggetti multi-paradigma [6]. Infatti, supporta il paradigma a oggetti, la programmazione strutturata e la programmazione funzionale. Uno dei motivi principali per cui Python è molto utilizzato in diversi settori dell'informatica è la possibilità di sviluppare applicazioni di scripting, distribuite e di computazione numerica. Grazie alla sua struttura molto flessibile, come per esempio la gestione dei tipi e il supporto alla programmazione funzionale, Python consente di gestire i dati con elevata dinamicità permettendo efficaci implementazioni di *Data Cleaning* con relativamente poche righe di codice. Per questo motivo, è uno dei linguaggi più utilizzati nella Data Science e più nello specifico, nel Data Mining.

Abbiamo scelto Python per questo progetto proprio per la sua possibilità di sviluppare efficientemente moduli come applicazioni di scripting utilizzando potenti librerie per elaborazioni di dati come *Pandas* e per la sua estrema flessibilità anche grazie ai metodi di programmazione funzionale che ci hanno permesso, usando un numero relativamente contenuto di righe di codice, lo sviluppo degli algoritmi di Record Linkage e Data Fusion di questa tesi.

Nello specifico, le librerie di Python più importanti per lo sviluppo di questo progetto sono *Pandas* e *Jellyfish*. Per comprendere l'importanza del Data Cleaning è rilevante citare anche altre librerie che hanno permesso di manipolare più efficacemente i dati. In particolare, ci riferiamo a *RecordLinkage*, *RdfLib*, *Read_pdf*, *Urllib*, *Json*, e *OrderedDict*.

5.2. Libreria Pandas

La libreria Pandas permette la manipolazione di dati sia in formato sequenziale che tabellare [7]. In particolare, fornisce la possibilità di:

- a. importare dati da file di diverso formato come *csv*, *xml*, *xls*, *txt* in DataFrame, una struttura in due dimensioni;
- b. eseguire operazioni di indicizzazione e aggregazione;
- c. eseguire operazioni statistiche;

- d. visualizzare i risultati.

I file di diverso formato sono trasformati in DataFrame tramite l'utilizzo di opportune funzioni di lettura del file sorgenti denominate *read()*. Durante il progetto abbiamo utilizzato le funzioni *read_csv()*, *read_excel()* e *read_json()*, che restituiscono un DataFrame. La prima per aprire i file in formato csv e in formato txt, la seconda per aprire i file in formato xml e xls. Inoltre, le funzioni *read()* di Pandas permettono la gestione di molte opzioni al fine manipolare al meglio di filtrare al meglio le informazioni. Per questo progetto sono state usate le opzioni di gestione dei separatori, di codifica e di indice di colonna. Quest'ultima opzione indica la colonna in cui sono contenuti gli indici o ne aggiunge una nuova.

Grazie all'entità DataFrame contenente il dataset di interesse suddiviso in classi denominate *Series*, come mostrato in Figura 8, abbiamo eliminato le barriere date dal diverso formato e abbiamo potuto gestire due dataset come due entità DataFrame che possono confrontarsi e scambiarsi tra loro informazioni.

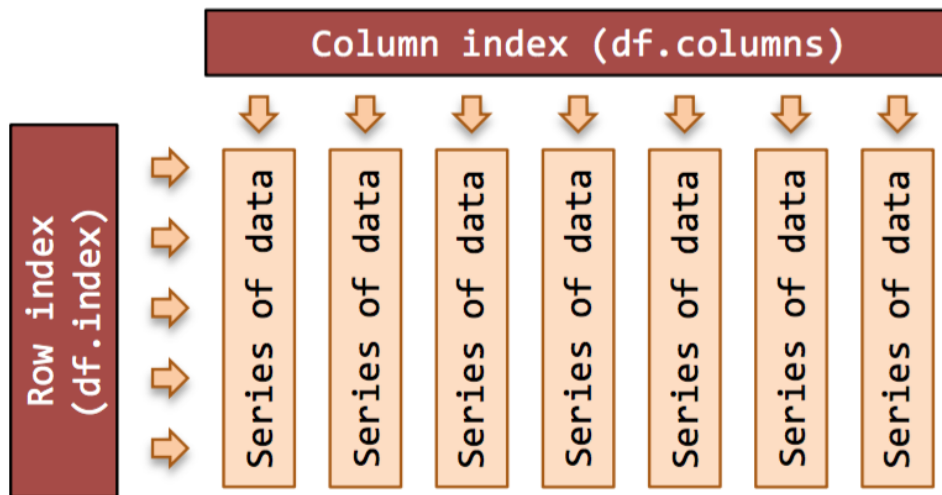


Figura 8: I DataFrame sono divisi in Series di dati

Inoltre, è possibile unire i dataset tra loro con i metodi *merge*, *join*, *concat* o *append* che forniscono lo stesso risultato delle tipiche clausole SQL.

Durante la tesi, una volta terminate le opportune manipolazioni che vedremo nel prossimo capitolo, abbiamo concatenato i DataFrame in un'unica entità per

mezzo della funzione *concat()*. La concatenazione permette di unire due dataset nel seguente modo:

- a. se è presente lo stesso nome dell'attributo nei due DataFrame, i valori di quell'attributo sono inseriti nella stessa colonna;
- b. in caso contrario, gli attributi presenti in uno solo dei dataset sono mantenuti nel DataFrame e una cella vuota è inserita nel campo del record del dataset che non contiene quell'attributo.

Infine, la funzione *concat* permette di individuare i record dei due dataset con l'opzione *keys*. Essa aggiunge una colonna in cui è possibile contraddistinguere i record dei dataset con un'opportuna etichetta.

5.3. Libreria Jellyfish

La libreria Jellyfish implementa in Python algoritmi di comparazione di stringhe e codifica di fonemi [8]. Nello specifico, la libreria include per:

- a) la comparazione di stringhe, gli algoritmi:
 - a. distanza di Levenshtein;
 - b. distanza di Damerau-Levenshtein;
 - c. distanza di Jaro;
 - d. distanza di Jaro-Winkler;
 - e. *Match Rating Approach Comparison*;
 - f. distanza di Hamming.
- b) La codifica di fonemi:
 - a. *American Soundex*;
 - b. *Metaphone*;
 - c. *NYSIIS (New York State Identification and Intelligence System)*;
 - d. *Match Rating Codex*.

Grazie agli metodi forniti dalla libreria, abbiamo potuto scegliere tra i migliori algoritmi di misurazione di stringhe così da avere a disposizione il migliore strumento che possa essere utile nel campo del Record Linkage. In particolare, la scelta ricade su un unico algoritmo di comparazione stringhe che possa permettere di individuare se due stringhe siano simili tra loro al fine di poterne scegliere soltanto una da mantenere nel dataset. Per questo motivo, abbiamo scartato inizialmente la *Match Rating Approach Comparison* in quanto più adatto al contesto dei fonemi che alla sintassi [27]. Tra i rimanenti, abbiamo notato che alcuni algoritmi sono una versione migliorata più recente. Di conseguenza, abbiamo deciso di scartare gli algoritmi di distanza di Levenshtein e di distanza di Jaro perché siamo in possesso di una loro versione migliore. Allo stesso modo, abbiamo scelto di scartare la distanza di Hamming in quanto la distanza di Levenshtein, che abbiamo scartato, è una sua versione migliorata perché permette, a differenza della distanza di Hamming, di gestire stringhe di diversa lunghezza. Infine, tra la distanza di Damerau-Levenshtein e la distanza di Jaro-Winkler, abbiamo scelto la seconda per due motivi. Il primo motivo è dato dal grande utilizzo che ha la distanza di Jaro-Winkler nel campo del Record Linkage. Il secondo motivo è dato dal risultato normalizzato tra zero e uno che è coerente con il resto dei metodi usati per questo progetto di tesi.

5.4. Librerie di Data Cleaning e Utility

Data l'importanza che riserva la Data Cleaning nel mondo della Data Science, riteniamo sia fondamentale scendere nel dettaglio del metodo *clean()* presente nella libreria Record Linkage che ci ha permesso di confrontare al meglio i valori presenti nei dataset [30].

```
recordlinkage.preprocessing.clean(s, lowercase=True,
replace_by_none='[^\\-\\_A-Za-z0-9]+', replace_by_whitespace='[\\-\\_]',
strip_accents=None, remove_brackets=True, encoding='utf-8',
decode_error='strict')
```

La funzione *clean* prende in ingresso una classe *pandas.Series*, *s* e ulteriori parametri opzionali con valori di default. Nello specifico, l'opzione *lowercase* converte le stringhe contenute in *s* in modo che tutti i caratteri siano minuscoli; *replace_by_none()* elimina tutti i caratteri presenti nella *regular expression* inserita, *replace_by_whitespace()* sostituisce l'espressione con uno spazio, *strips_accents()* rimuove gli accenti durante la fase di *pre-process*, *encoding* indica la codifica usata e *decode_error()* gestisce gli errori in fase di decodifica con la possibilità di scegliere tra *strict*, che ritorna un *UnicodeDecodeError*, *ignore* e *replace*.

Nonostante *Pandas* fornisca una struttura comune per dataset in differenti formati, *DataFrame* è particolarmente usata perché permette di aprire i comunissimi file CSV, JSON e i formati di Excel. D'altro canto, non fornisce attualmente la possibilità di importare direttamente i formati RDF e PDF. Per questo, abbiamo usato due librerie ad hoc che ne hanno permesso l'importazione parziale: *Rdflib* e *Read_pdf*.

Infine, le funzioni di Utilità di maggiore importanza per lo sviluppo del progetto che meritano di essere citate sono *Urllib* e *OrderedDict*. Il primo permette di lavorare con gli *URL* e ci ha permesso di interfacciarci con le API del portale dati.gov.it per poter ricercare all'interno della base di dati [31] con l'aiuto della libreria *Json*. Il secondo fornisce un dizionario ordinabile che è stato estremamente utile quando abbiamo avuto l'esigenza di contare e successivamente ordinare per occorrenze le parole presenti nei dataset usati [32]. Senza la classe *OrderedDict* sarebbe stato molto più complicato sviluppare l'algoritmo fondamentale del metodo *record_linkage()* che vedremo nel prossimo capitolo.

6. Implementazione

In questo capitolo, mostriamo il lavoro svolto e spieghiamo le scelte realizzative. La libreria *opendatagov* è formata da sette funzioni principali che, come mostrato in Figura 9, definiscono un completo processo di estrazione, pulizia, trasformazioni e caricamento dei dati dal portale dati.gov.it. In particolare, nel paragrafo 6.1, concentriamo la nostra attenzione sulla funzione di ricerca tramite Web API *search()*. Nel paragrafo 6.2, mostriamo la funzione *ext()* di trasformazione dei dati di diverso formato nella struttura unica DataFrame. Nel paragrafo 6.3, definiamo le pratiche di Data Cleaning ottenute con lo sviluppo della funzione *preprocess*. La parte centrale del progetto è concentrata nei paragrafi 6.4, 6.5 e 6.6 dove sono spiegati, rispettivamente, la funzione di *header_linkage()* per trovare la similarità tra i nomi degli attributi, la funzione *find_claims()*, che ha lo scopo di cercare le parole comuni più usate nei due dataset e *record_linkage()*, che sfrutta l'algoritmo implementato in *find_claims()* per trovare le similarità tra le parole presenti negli attributi dei due dataset. Concludiamo il capitolo con il paragrafo 6.7 dove mostriamo la funzione *merge_keys()*, che gestisce le scelte di priorità tra le funzioni *header_linkage()* e *record_linkage()*.

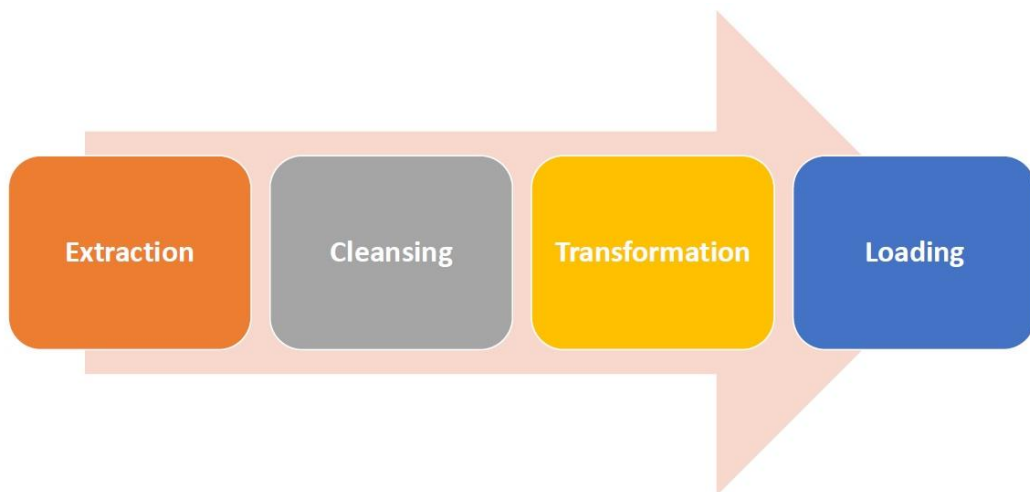


Figura 9: fasi del processo ETL

6.1. Fase di Ricerca

La funzione di ricerca *def search(key, kind)* sfrutta le Web API del portale dati.gov.it per permettere agli utenti una ricerca all'interno del database governativo. Abbiamo ritenuto importante aggiungere una libreria di supporto come *search()* in modo da fornire un ampio spettro di strumenti utili per lavorare sul portale.

Per questa funzione abbiamo sfruttato due web API della piattaforma di metadata CKAN, *package_list* e *package_show*. Come indicato sulla documentazione del portale dati.gov.it [33], il primo metodo: “*permette l'interrogazione dell'indice di tutti i record.*”, mentre il secondo metodo permette di: “*fare interrogazioni puntuali ai singoli record con la chiamata.*”. Sono state gestite le risposte e fornite come valori di ritorno dalla funzione. Nello specifico, l'utente inserisce come parametri d'ingresso della funzione una stringa *key* contenente le parole chiavi da ricercare all'interno del database e il carattere *kind* che può solo avere valore “L” per invocare il metodo *package_list* o “S” per invocare il metodo *package_show*.

6.2. Fase di Estrazione

La funzione *def ext(url, index=None)* controlla il formato del file e crea un'entità DataFrame. Il metodo prende in ingresso due parametri, *url* e *index*. Il primo parametro è una stringa contenente l'indirizzo, assoluto in caso di utilizzo di una fonte online come il portale dati.gov.it o eventualmente relativo in caso di uso di file locali. Il secondo parametro è un valore intero o *None*, impostato come valore di default, che indica la colonna indice del dataset. L'utente può scegliere la colonna indice oppure usare lasciare inalterato il parametro di default *None*, che automatizza il processo aggiungendo una nuova colonna indice nella prima posizione del dataset.

Il metodo verifica il tipo di formato controllando i caratteri successivi all'ultimo punto nella stringa d'ingresso *url* e intercetta i formati *csv*, *json*, *xml*, *xls*,

rdf, *pdf* e *txt*. Considerando che il portale dati.gov.it contiene soprattutto file in formato *csv*, abbiamo deciso di gestire e risolvere approfonditamente questo formato. In particolare, la funzione prova inizialmente ad aprire i file *csv* utilizzando la funzione *read_csv()* di Pandas impostando il separatore standard “punto e virgola” e sfruttando l’engine “python”. Durante l’esecuzione dei test, il metodo ha innescato tre eccezioni che abbiamo gestito:

- a. *UnicodeDecodeError* e b. *UnicodeEncodeError*, che indicano un problema nella codifica dei caratteri. È un problema noto nel mondo della Data Science e strettamente dipendente dalle scelte e dagli strumenti usati dalla fonte. Abbiamo trovato una soluzione richiamando nuovamente la funzione *read_csv()* con valore del parametro *encoding*, di default *None*, come *latin-1*. La scelta è frutto di una ricerca empirica che ci ha mostrato come la codifica latin-1 sia valida per la maggior parte dei casi presi in considerazione.
- c. *Pandas.errors.ParserError*, indica un problema nel separatore utilizzato. Abbiamo gestito il problema con l’utilizzo di un’opportuna variabile indicante che il DataFrame, se creato, è vuoto e bisogna procedere a ulteriori tentativi con altri separatori. Nello specifico, abbiamo gestito i separatori standard per *csv*: “virgola”, “tabulatore” e “spazio”.

Un ulteriore problema è la generazione di una variabile DataFrame vuota che non innesca alcuna eccezione. È solitamente dovuta a un’incorretta gestione dei separatori. Tale condizione è indicata tramite l’utilizzo di una variabile. Inoltre, è possibile che la funzione rilevi che ci siano più separatori validi per l’apertura del file. In questo caso, l’utente deve scegliere tra le possibili soluzioni valide.

Infine, nel caso in cui il formato intercettato non sia tra quelli disponibili, il metodo chiude l’applicazione spiegandone i motivi.

6.3. Fase di Data Cleaning

La funzione *def preprocess(df, th_jaro, user)* ha il compito di ripulire i valori degli attributi di un dataset al fine di facilitare le successive operazioni di controllo di similarità tra stringhe. Infatti, il metodo è richiamato all'interno di tutte le funzioni principali del progetto per migliorare sensibilmente la qualità del risultato finale.

La funzione prende in ingresso tre parametri. Il primo è il dataset sotto forma di DataFrame. Il secondo è un valore numerico decimale corrispondente al valore di soglia utilizzato per l'algoritmo di misurazione di similarità tra stringhe Jaro-Winkler. Il terzo parametro di ingresso è una variabile booleana che indica se l'utente vuole automatizzare il processo di scelta delle parole da mantenere all'interno del dataset in caso di conflitto, nel caso che l'opzione sia disattivata, oppure se vuole scegliere personalmente, nel caso l'opzione sia abilitata. Il risultato delle funzioni è la modifica del DataFrame corrispondente al dataset.

Il metodo è suddiviso in tre fasi principali:

- a. Rimozione di tutti i caratteri non necessari, e spesso dannosi, per il confronto tra stringhe;
- b. per ogni valore all'interno di tutti gli attributi del dataset, individuazione per mezzo dell'algoritmo Jaro-Winkler di stringhe simili tra loro;
- c. gestione del conflitto tra parole simili con la possibilità per l'utente di scegliere quali parole mantenere e quali modificare all'interno del dataset.

Il punto *a* è il garante per qualità del risultato finale. Per la modifica dei record abbiamo scelto la funzione *clean()* presente nella libreria RecordLinkage a cui segue un controllo degli ultimi caratteri del valore dell'attributo con lo scopo di rimuovere eventuali caratteri di *spazio* multipli superflui e potenzialmente dannosi nell'individuazione di similarità tra stringhe.

Nel punto *b*, ogni singolo record ora ripulito, è confrontato con tutti gli altri ad eccezione dei valori numerici. Infatti, in tutti i contesti e in particolare nel mondo della Scienza dei Dati, vi è una profonda differenza semantica tra i valori numerici e testuali. Sarebbe errato l'utilizzo di un'unica soluzione per entrambi i tipi di valori perché danneggerebbe il risultato di entrambi i campi di lavoro. Per questo progetto abbiamo scelto di concentrarci soltanto sulle parole con la conseguente scelta, in questa fase, di utilizzare l'algoritmo Jaro-Winkler. Nello specifico, verifichiamo se esistono parole che abbiano una similarità che sia almeno uguale al valore di soglia scelto come parametro di ingresso, escludendo ovviamente le parole con similarità uno, cioè uguali. Questa fase è molto delicata, perché la scelta del valore di soglia dell'algoritmo Jaro-Winkler modifica sostanzialmente il risultato finale e spesso è necessario scegliere il valore di soglia solo dopo un accurato studio preliminare.

Infine, il punto *c* prevede una gestione del conflitto che può essere automatizzata o manuale. Nel caso in cui l'opzione *user* sia abilitata, l'utente riceve a video una lista di coppie di parole simili tra loro di cui ne sceglie una che è mantenuta all'interno del dataset e che sostituisce quella scartata. Le possibili scelte dell'utente sono:

- a. il carattere "1", per scegliere la prima parola;
- b. il carattere "2", per scegliere la seconda parola;
- c. il carattere "0", per saltare la scelta. In questo caso, non è apportata alcuna modifica.

Se l'opzione *user* è disabilitata, il sistema sceglie la prima parola.

6.4. Fase di Text Mining

L'algoritmo centrale del progetto di tesi risiede nella funzione *def find_claims(df, threshold, stopword, th_jaro, user)*. Essa prende in ingresso cinque parametri. In ordine:

- a. il dataset di utilizzo, sottoforma di DataFrame;
- b. il valore di soglia delle occorrenze, un valore numerico decimale corrispondente al limite inferiore per cui una parola compare sufficientemente volte all'interno di una colonna attributo per essere considerata idonea;
- c. le stopwords, un vettore di stringhe contenenti parole da scartare in quanto fuorvianti per l'algoritmo;
- d. il valore di soglia dell'algoritmo Jaro-Winkler, un valore numerico decimale necessario per la fase di Data Cleaning interna alla funzione.
- e. Opzione *user*, una variabile booleana obbligatoria per la fase di Data Cleaning interna alla funzione.

Il risultato finale è contenuto in un vettore di dizionari ordinati che contengono le occorrenze delle parole di ogni attributo. Il metodo è diviso in tre frasi principali: fase di Data Cleaning, fase di conteggio delle occorrenze e fase di normalizzazione e scelta.

Per la prima fase, abbiamo sfruttato la funzione già precedentemente implementata *preprocess()* per ripulire tutti i record degli attributi. Per la seconda fase, abbiamo diviso ogni record in singole parole sfruttando gli spazi presenti all'interno di ogni valore. Ogni parola è stata inserita all'interno del dizionario del proprio attributo con ordinamento decrescente per occorrenza di ogni parola. Per la terza fase, abbiamo normalizzato le occorrenze di ogni parola per grandezza della colonna attributo, non considerando i record vuoti o non definiti. Successivamente, abbiamo filtrato questo valore scartando:

- a. Singoli caratteri;
- b. valori numerici;
- c. stopwords;
- d. valori normalizzati inferiori al valore di soglia impostato dall'utente.

La funzione restituisce il risultato di quest'ultima operazione.

6.5. Fase di Record Linkage

Abbiamo diviso la fase di record linkage di questo progetto di tesi in due parti principali. La prima parte riguarda la misurazione di differenza tra stringhe dei nomi degli attributi. Il risultato di questa fase risiede nella funzione *header_linkage()* esposta nel sotto-paragrafo 6.5.1. La seconda parte, spiegata nel sotto-paragrafo, 6.5.2, sfrutta l'algoritmo di *text mining* implementato e consiste nel verificare se due attributi, indipendentemente dal loro nome, contengono parole uguali ripetute un numero sufficiente di volte per cui si possa dedurre che i due attributi dei due dataset si riferiscano allo stesso contesto.

6.5.1. Misura di Differenza tra Stringhe nel Nome degli Attributi

Il metodo *def header_linkage(left_df, right_df, treshold = 0.8)* confronta i nomi di colonna attributo di due dataset al fine di indicare all'utente che tale similarità può essere presente anche nel contenuto degli attributi stessi. La funzione richiede in ingresso tre parametri. I primi due parametri sono di tipo DataFrame e corrispondono ai dataset di utilizzo. Il terzo parametro è il valore di soglia dell'algoritmo Jaro-Winkler, con valore di default pari a 0.8.

Header_linkage() è divisa in due step principali: data cleaning e misura di differenza tra stringhe. Nel primo passo, abbiamo ripulito le colonne dei due dataset con la funzione *clean()* della libreria Record Linkage. Nel secondo passo, abbiamo sfruttato nuovamente l'algoritmo Jaro-Winkler presente nella libreria Jellyfish per trovare stringhe simili tra i nomi degli attributi con limite inferiore pari al valore di soglia scelto dall'utente.

6.5.2. Record Linkage sui Valori degli Attributi

Il metodo `def record_linkage(left_df, right_df, left_threshold = 0, right_threshold = 0, th_jaro = 0.944, user = True)` verifica l'esistenza di parole in comune tra i dataset e, se la loro presenza è significativa, ne calcola il peso.

La funzione prende in ingresso sei parametri. I primi due, obbligatori, sono i dataset sottoforma di DataFrame da prendere in considerazione. I restanti hanno valori di default che ne permettono l'omissione e servono per specificare le opzioni della funzione `find_claims()`, che è stata usata all'interno di questo metodo. Nello specifico, il terzo e quarto parametro di ingresso, `left_threshold` e `right_threshold`, sono dei valori di soglia che filtrano il peso delle parole, inteso come occorrenza all'interno di ogni attributo. Di default sono impostati a zero allo scopo di non compiere alcuna operazione di filtraggio. Il quinto parametro è il valore di soglia dell'algoritmo Jaro-Winkler. Le ricerche empiriche hanno mostrato come un valore vicino a 0.95 riesca a essere un buon compromesso per un buon numero di dataset. Tuttavia, è necessaria un'accurata scelta per ogni caso. Infine, come abbiamo già visto, l'ultimo parametro abilita la gestione manuale della scelta delle parole simili. Il risultato del metodo è formato da coppie di nomi attributo che possono avere una correlazione, perché contengono parole in comune che hanno un'occorrenza tale in entrambe le sorgenti da poter stabilire una correlazione tra gli attributi.

Abbiamo diviso la funzione in cinque parti fondamentali:

- a. Invocazione del metodo `find_claims()` per entrambi i dataset di utilizzo e ricerca delle parole in comune sulle colonne attributo di ogni dataset;
- b. calcolo del peso totale delle parole rispetto ai due dataset come somma del peso che ogni parola ha all'interno di ognuno di essi;
- c. creazione per ogni parola trovata di una struttura formata da quattro valori: parola in comune, nome dell'attributo in cui è presente nel primo dataset e nel secondo dataset, valore numerico che indica il peso totale della parola in comune normalizzato rispetto ai due dataset considerati in un intervallo da zero e due.
- d. ordinamento della struttura per peso totale;

- e. creazione di coppie di nomi attributo del primo e del secondo dataset formate a partire dai migliori *match* ottenuti con il calcolo del peso delle parole in comune.

6.5.3. Fase di Fusione dei Risultati

Le due funzioni *header_linkage()* e *record_linkage()* forniscono due valutazioni diverse. Di conseguenza, possono essere usate separatamente oppure fornire un risultato totale più ampio fondendo i valori trovati.

Il metodo *def merge_keys(left_keys, left_records, right_keys, right_records)* unisce due coppie di vettori di stringhe. Prende in ingresso quattro vettori di stringhe che contengono, rispettivamente:

- a. Il primo e il terzo parametro contengono i nomi degli attributi, rispettivamente, del primo e del secondo dataset trovati dalla funzione *header_linkage()* che si consiglia di concatenare;
- b. Il secondo e quarto parametro contengono i valori degli attributi, rispettivamente, del primo e del secondo dataset trovati dalla funzione *record_linkage()* che si consiglia di concatenare;

Il risultato è l'unione dei risultati dei due metodi con lo scopo di dare priorità, in caso di conflitto, alle correlazioni trovate nei nomi degli attributi.

7. Risultati Sperimentali

Abbiamo testato la libreria sviluppata ricercando due argomenti di interesse nazionale al fine di simulare una reale ricerca di un utente. In questo capitolo mostriamo le difficoltà che si possono trovare in una prima ricerca sul portale dati.gov.it e la risoluzione di alcune problematiche per mezzo del progetto di tesi sviluppato. In particolare, nel paragrafo 7.1, spieghiamo i passaggi generali utilizzati per l'implementazione dei test, che sono utilizzati da entrambi le prove svolte.

Abbiamo effettuato il primo test, spiegato nel paragrafo 7.2, cercando sulla piattaforma governativa i dataset dei beni confiscati alla criminalità organizzata. Nello specifico, ci siamo concentrati su due dataset, il primo dei beni confiscati alla mafia nella città di Milano e il secondo riguardante la città di Catania.

Il secondo test, mostrato nel paragrafo 7.3, è incentrato sul patrimonio culturale italiano. In particolare, i due dataset che abbiamo usato riguardano l'anagrafica dei musei della regione Lazio e della regione Sicilia.

7.1. Metodologia di Test

Lo scopo delle simulazioni di ricerca è estrarre informazioni da più sorgenti e concatenarle in un'unica fonte al fine di poter fornire un unico documento che possa essere successivamente manipolato per estrarre e confrontare un numero maggiore di informazione di migliore qualità.

Abbiamo diviso il nostro esperimento nei seguenti step:

- a. Ricerca sul portale dati.gov.it con chiave specifica;
- b. scelta di due dataset con sufficienti informazioni utili;
- c. generazione dei DataFrame e salvataggio in locale dei file;
- d. analisi dei nomi degli attributi dei dataset per trovare delle correlazioni;

- e. analisi dei valori degli attributi dei dataset per trovare correlazione tra le parole contenute nei *record*;
- f. salvataggio in locale dei dataset ripuliti dalle operazioni di Data Cleaning;
- g. fusione delle due analisi al punto *d* e al punto *e* al fine di ottenere informazioni complete sulle scelte da attuare;
- h. concatenazione dei due dataset in un unico DataFrame e salvataggio del corrispondente file in locale.

7.2. Test su Beni Confiscati alla Mafia

Il nostro primo passo è cercare all'interno del portale dati.gov.it. La chiave di ricerca è: "mafia". Nel momento in cui è stato effettuato il test, il portale mostrava quattro *package*. Dopo un primo controllo preliminare puramente esplorativo, abbiamo deciso di usare i due dataset che forniscono più informazioni. I package hanno rispettivamente titolo: "*sociale-beni-confiscati-mafia*" per il dataset fornito dal comune di Milano [34] ed "*elenco-beni-confiscati-mafia*" per il dataset fornito dal comune di Catania [35]. Per semplicità e omogeneità con il nome dei parametri della funzione di concatenazione, denominiamo i dataset con gli stessi nomi usati all'interno del progetto. Nello specifico, per il dataset del Comune di Milano usiamo *mafia_left*, mentre *mafia_right* per quello del Comune di Catania. I dataset così ottenuti possono essere richiamati ogni volta dal web indicando l'opportuno indirizzo oppure possono essere scaricati per un caricamento da locale. È opportuno specificare che gli Open Data sono soggetti ad aggiornamenti e modifiche che possono minare la riusabilità del lavoro prodotto qualora i dataset ricevano cambiamenti sostanziali alla struttura.

Il dataset *mafia_left* è composto da otto attributi e centocinquantacinque *record*:

- a. Il nome del primo attributo, "*n. unitÃ*", soffre di problemi di codifica e contiene dei valori numerici di difficile interpretazione;

- b. il secondo attributo, “*tipologia*”, indica per l’appunto la tipologia dell’immobile confiscato. Nello specifico, i valori degli attributi sono stringhe di cui possiamo notare degli errori di ortografia come “*localecommerciale*” e la sua corretta forma “*locale commerciale*”;
- c. il terzo attributo, “*consistenza*” contiene la grandezza del bene confiscato in termini di metri quadri in alcuni casi e di vani in altri. Nonostante l’ovvia ingente presenza di valori numerici, i *record* sono delle stringhe;
- d. il quarto attributo, “*ubicazione*” indica l’indirizzo dell’immobile sotto forma di stringhe;
- e. il quinto attributo, “*concessionari*” specifica le società, cooperative o associazioni a cui è concessa la gestione del bene confiscato sotto forma di stringhe. Infatti, la *legge n. 109/96 per il riutilizzo pubblico e sociale dei beni confiscati alle mafie* specifica che un bene confiscato dallo Stato alla criminalità organizzata può essere concesso a un “*amministratore*” che gestisca il riutilizzo dell’immobile;
- f. il sesto attributo, “*durata anni*” contiene valori numerici interi, stringhe e campi vuoti. Il suo significato deve essere interpretato, ma l’ipotesi più plausibile è che sia legato all’attributo *concessionari* e specifichi la durata degli anni della concessione;
- g. il settimo e ultimo attributo, “*progetto*”, è una stringa uguale per tutti i *record* con valore: “*Progetto Sociale*”.

Il dataset *mafia_right* è composto da dieci attributi e trecentocinquantanove *record*:

- a. Il primo attributo, “*numero*”, è composto da soli valori numerici interi e un campo vuoto. Riteniamo i valori di questo attributo corrispondano all’indice del dataset considerando che ogni record differisce dal precedente di dieci unità;
- b. il secondo attributo, “*immobile*” contiene stringhe che specificano l’indirizzo dell’immobile;

- c. il terzo attributo, “*Decreto Ministeriale*” è formato da valori numerici interi che come suggerisce il nome dell’attributo indicano probabilmente il numero di protocollo del decreto che ha ufficializzato la concessione;
- d. il quarto attributo, “*Data*”, può avere diversi significati. L’interpretazione più plausibile è che sia legato all’attributo precedente “*Decreto ministeriale*” e di conseguenza specifica la data del decreto stesso. Le date possono essere trattate in diverso modo grazie ad alcuni strumenti della libreria Pandas. Nel nostro caso considereremo i valori di questo attributo come stringhe;
- e. il quinto, sesto e settimo attributo, rispettivamente “*Foglio*”, “*Particella*” e “*Sub*”, sono dei valori numerici non definibili in un campo semantico preciso. Probabilmente sono legate a un ordinamento protocollare che non possiamo decifrare con le informazioni in nostro possesso;
- f. l’ottavo attributo, “*Tipologia*”, è composto da stringhe che mostrano il tipo di immobile confiscato;
- g. il nono attributo, “*Associazione assegnataria*”, è formato da stringhe e campi vuoti che indicano la società, associazione e cooperativa che amministra il bene confiscato;
- h. il decimo attributo, “*Note*”, è composto da stringhe con informazioni di vario tipo. Particolarmente interessante è la presenza in alcuni record indicanti il numero di vani di cui è composto il bene immobile.

Dalla prima analisi dei dataset notiamo come le due sorgenti forniscano le stesse informazioni in alcuni attributi che hanno nomi differenti. Di conseguenza, riteniamo questo scenario adatto per provare la libreria sviluppata con questo progetto di tesi. Nello specifico, ci aspettiamo di:

- a. Individuare i nomi degli attributi molto simili e che contengono la stessa informazione, come si può notare dalla Tabella 4. Nel nostro

caso ci riferiamo agli attributi, rispettivamente di *mafia_left* e *mafia_right*, “tipologia” e “Tipologia”;

- b. associare gli attributi formati da stringhe che contengono lo stesso valore semantico. Ci riferiamo alle coppie, rispettivamente di *left_mafia* e *right_mafia*: “ubicazione” – “Immobile” e “concessionari” – “Associazione Assegnataria”.

n. unitÃ	tipologia	ubicazione	Immobile	Tipologia
1	abitazione	Via ArquÃ	Via Gambetta n 46	villetta
2	abitazione	Via Arzaga	Via Nazario Sauro ang V D Chiesa	appartamento
3	abitazione	Via Arzaga	Via Nazario Sauro ang V D Chiesa	appartamento
4	box	Via Arzaga	Via Nazario Sauro ang V D Chiesa	appartamento
5	box	Via Asiago	Via Randazzo 27	appartamento
6	abitazione	Via Balducci	Via Randazzo 27	giardino
7	abitazione	Via Baldo degli Ubaldi	Via Masaniello 4/a	bottega
8	abitazione	Via Bassano del Grappa	Via Anapo 16 â€“ 18	magazzino
9	abitazione	Via Bembo	Via Anapo 16 â€“ 18	magazzino

Tabella 4: contenuto parziale dei dataset, a sinistra, *mafia_left* e, a destra, *mafia_right*.

Il secondo passo è sfruttare i DataFrame ottenuti per cercare similarità tra i nomi degli attributi. Di conseguenza, invochiamo con valori di default la funzione *header_linkage()* che effettua prima l’operazione di Data Cleaning sui nomi degli attributi e dopo li confronta. Notiamo che:

- a. Sono rimossi tutti i caratteri maiuscoli dai nomi attributo. Di conseguenza, ora due nomi attributo sono esattamente uguali: “tipologia” – “tipologia”;
- b. l’algoritmo di *pre-process* non intercetta l’errore di codifica “n. unitÃ”.

Il risultato della funzione è il consiglio di concatenare gli attributi con nome “tipologia”.

Il terzo passo è di usare la funzione *record_linkage()* con valori di default e l'opzione User abilitata. In modo analogo al passo precedente, la funzione ripulisce i record degli attributi, salva i dataset ripuliti con un nuovo nome per eventuali usi futuri e richiede all'utente di risolvere il conflitto tra le parole “*locale commerciale*” – “*localecommerciale*” e “*abitazione e solaio di pertinenza*” – “*abitazione e solai di pertinenza*”. Successivamente, fornisce i valori di similarità tra gli attributi basata sulle parole in comune. Come si può notare dall'output del progetto di tesi mostrato sotto, la funzione *record_linkage()* trova molte parole chiave in comune, di cui quattro permettono l'associazione degli attributi:

- a. “Immobile” e “ubicazione” per mezzo della parola “*via*” con peso rispettivamente “1.0” e “0.93”;
- b. “Tipologia” e “tipologia” per mezzo della parola “*terreno*” con peso rispettivamente “0.13” e “0.01”;
- c. “associazione assegnataria” e “concessionari” per mezzo della parola “*centro*” con peso rispettivamente “0.27” e “0.04”;
- d. “note” e “consistenza” per mezzo della parola “*vani*” con peso rispettivamente “0.08” e “0.51”.

I pesi della parola in comune sono sommati ottenendo la struttura decisionale, mostrata in Tabella 5, che fornisce i consigli di associazione. Da essa sono estratti solo le migliori associazioni per singolo nome attributo. Di conseguenza anche un peso totale abbastanza basso come “0,14” di “tipologia” – “tipologia” è considerato un buon consiglio rispetto allo “0,53” di “tipologia” – “consistenza” perché l'attributo “consistenza” ha già un miglior *match* con “note” dal valore di “0,59”.

The word "via" stay in the attribute column of the left Dataset: immobile with weight: 1.0 and in the attribute column of the right Dataset: ubicazione with weight: 0.9294871794871795.

[...]

The word "terreno" stay in the attribute column of the left Dataset: tipologia with weight: 0.13157894736842105 and in the attribute column of the right Dataset: tipologia with weight: 0.00641025641025641.

[...]

The word "centro" stay in the attribute column of the left Dataset: associazione assegnataria with weight: 0.2692307692307692 and in the attribute column of the right Dataset: concessionari with weight: 0.038461538461538464.

[...]

The word "vani" stay in the attribute column of the left Dataset: note with weight: 0.07894736842105263 and in the attribute column of the right Dataset: consistenza with weight: 0.5128205128205128.

Attribute names found in record linkage are:

['immobile', 'note', 'associazione assegnataria', 'tipologia'] ['ubicazione', 'consistenza', 'concessionari', 'tipologia']

Claim	Left Attribute Name	Right Attribute Name	Weight
via	immobile	ubicazione	1,93
via	immobile	concessionari	1,18
vani	immobile	consistenza	0,64
vani	note	consistenza	0,59
mq	immobile	consistenza	0,53
mq	tipologia	consistenza	0,53
centro	associazione assegnataria	concessionari	0,31
onlus	associazione assegnataria	concessionari	0,19
terreno	tipologia	consistenza	0,17
villetta	tipologia	tipologia	0,14
terreno	tipologia	tipologia	0,14
magazzino	tipologia	tipologia	0,14
casa	associazione assegnataria	concessionari	0,13
per	associazione assegnataria	concessionari	0,10

monte	immobile	ubicazione	0,09
libera	associazione assegnataria	concessionari	0,09
aiuto	associazione assegnataria	concessionari	0,05
vita	associazione assegnataria	concessionari	0,05
dei	immobile	concessionari	0,03

Tabella 5: struttura decisionale di record_linkage() per il test Mafia

Il quarto passo e ultimo passo è la fusione dei consigli delle due funzioni e la concatenazione dei dataset. È interessante notare come l’associazione per nomi attributo “tipologia” – “tipologia” di *header_linkage()* sia presente anche nelle associazioni di *record_linkage()*.

*{'tipologia': 'tipologia', 'immobile': 'ubicazione', 'note': 'consistenza',
'associazione assegnataria': 'concessionari' }*

Infine, le associazioni sono fornite in output dal programma ed è creato un nuovo dataset con la concatenazione di *mafia_left* e *mafia_right*. Esso ha un’ulteriore colonna in cui è indicata la provenienza di ogni record, rispettivamente, con i label “Catania” e “Milano”.

7.3. Test su Anagrafica Museale

Abbiamo deciso di usare come chiave di ricerca per questo test la parola “musei” e abbiamo ottenuto in risposta sessantacinque dataset. Tra questi, la scelta è ricaduta su “*elenco-dei-musei-lazio*” e “*musei-gallerie-siti-archeologici*” che contengono, rispettivamente, le anagrafiche dei musei della regione Lazio e della regione Sicilia. Tra i formati disponibili abbiamo scelto i file in formato *csv*. Successivamente, come già fatto per il test precedente, per omogeneità dei nomi, indichiamo il dataset con i dati della regione Lazio come *musei_left* mentre *musei_right* per quelli della regione Sicilia, salvando poi i file in locale.

Q1.1_SA_DENOMINAZIONE_MUSEO_ISTITUTO	Q1.2_SA_INDIRIZZO_MUSEO_ISTITUTO	Q1.3_SA_LOCALITA_MUSEO_ISTITUTO
museo civico FERRANTE RITTATORE VONWILLER	VIA COLLE SAN MARTINO; 16	Farnese
ECOMUSEO DEL LITORALE ROMANO	VIA DEL FOSSO DI DRAGONCELLO; 172	LONGARINA/OSTIA
Abbazia di Trisulti	Via Trisulti; 8	-
MUSEO STORICO DELLA MOTORIZZAZIONE MILITARE DELL'ESERCITO	VIALE DELL'ESERCITO; 170	CECCHIGNOLA
MUSEO STORICO DELLA COMUNICAZIONE	VIALE EUROPA 190	-
Complesso di Capo di Bove	Via Appia Antica; 222	-
MUSEI CAPITOLINI	PIAZZA DEL CAMPIDOGLIO; 1	
MUSEO TEATRALE SIAE DEL BURCARDO	VIA DEL SUDARIO; 44	-
MUSEO CRIMINOLOGICO	VIA DEL GONFALONE; 29	-
Museo d'arte sacra di San Giovanni dei Fiorentini	via degli Acciaioli 2	
Necropoli della Banditaccia	Piazzale Mario Moretti (già della Necropoli)	-
CIVICO MUSEO D'ARTE MODERNA E CONTEMPORANEA	PIAZZA SANTA VITTORIA; 2	Anticoli Corrado
Complesso Archeologico di Malborghetto	Via Barlassina; 1	Via Flaminia; Km 19;00

Tabella 6: contenuto parziale del dataset musei_left

Durante la fase di estrazione di questa prova, abbiamo avuto modo di testare anche la gestione degli errori. Infatti, Pandas ha riconosciuto due separatori come validi: la *virgola* e il *tabulatore*. Simulando le operazioni di un utente, abbiamo aperto manualmente i file e notato come i dataset siano stati pensati per aver come separatore la virgola che è stata scelta come opzione tra quelle disponibili. Inoltre, i dataset non contengono un attributo di indice che è stata inserito automaticamente dal sistema con nome “*Column1*” in entrambe le sorgenti.

Il dataset *musei_left* è formato da venti attributi con nomi poco leggibili da un sistema informatico e trecentocinquantotto *record*. Nello specifico, forniamo sotto la lista di tutti gli attributi con descrizione di solamente quelli rilevanti per il test effettuato:

- a. “*ColumnI*”, valori numerici interi che indicano la colonna indice del dataset generata dal sistema;
- b. “*Comune*”, stringhe che indicano il comune del museo;
- c. “*Provincia*”, stringhe che indicano la provincia del museo”;
- d. “*Statale*” e “*Non statale*”, i valori possibili sono le stringhe “*Non MIBACT*” e “*MIBACT*”, acronimo di *Ministero per i beni e le attività culturali*;
- e. “*OMR*”, i valori possibili sono “*OMR*”, acronimo di *Organizzazione Museale Regionale*, oppure un campo vuoto;
- f. “*Sistema museale territoriale*”;
- g. “*Sistema museale urbano*”, stringhe o campi vuoti che indicano il sistema museale urbano.;
- h. “*Sistema museale tematico*”;
- i. “*Casa museo*”;
- j. “*Q1.1_SA_DENOMINAZIONE_MUSEO_ISTITUTO*”, stringhe che indicano il nome del museo;
- k. “*Q1.2_SA_INDIRIZZO_MUSEO_ISTITUTO*”, stringhe indicano l’indirizzo del museo;
- l. “*Q1.3_SA_LOCALITA_MUSEO_ISTITUTO*”;
- m. “*Q1.4_SA_UBICAZIONE/EDIFICIO_MUSEO_ISTITUTO*”;
- n. “*Cap*”;
- o. “*Q1.8_SA_TELEFONO_MUSEO_ISTITUTO*”, stringhe o campi vuoti che indicano il numero di telefono del museo. Data la lunghezza del valore numerico e l’eventuale presenza di caratteri speciali di delimitazione come trattini o spazi, riteniamo che i valori dell’attributo debbano essere gestiti come stringhe;
- p. “*Q1.9_SA_FAX_MUSEO_ISTITUTO*”;
- q. “*Q1.10_SA_EMAIL_MUSEO_ISTITUTO*”;
- r. “*Q1.11_SA_SITO WEB_MUSEO_ISTITUTO*”;
- s. “*latitudine*”;
- t. “*longitudine*”.

I record del dataset non presentano particolari errori di codifica e sono sufficientemente leggibili e comprensibili sia da un utente che da un sistema informatico, come si può notare nella Tabella 6.

Il dataset *musei_right* è formato da dodici attributi i cui nomi sono comprensibili tanto da un utente quanto da un sistema informatico e centotrenta *record*. Come per il precedente dataset, mostriamo sotto i nomi di tutti gli attributi dando una descrizione soltanto per quelli utili ai fini della comprensione della prova svolta:

- a. “*Column1*”, valori numerici interi che indicano la colonna indice del dataset generata dal sistema;
- b. “*Categoria*”, stringhe che indicano il tipo di area culturale;
- c. “*Provincia*”, stringhe che indicano la provincia del museo;
- d. “*Comune*”, stringhe che indicano il comune del museo;
- e. “*Denominazione*”, stringhe che indicano il nome del museo;
- f. “*Indirizzo*”, stringhe che indicano l’indirizzo del museo;
- g. “*Telefono*”, stringhe che indicano il numero di telefono del museo;
- h. “*Orari*”;
- i. “*Biglietto intero*”;
- j. “*Biglietto ridotto*”;
- k. “*Note*”;
- l. “*Scheda*”;

Come si può notare dalla Tabella 7, i valori degli attributi di *musei_right* sono ben strutturati, ma presentano alcuni problemi di codifica.

Da una prima analisi dei soli nomi degli attributi, possiamo notare come molti nomi facilmente interpretabili dalla mente umana, siano molto più difficili da intercettare da un sistema informatico a causa della scelta di sostituire in *musei_left* gli spazi con gli *underscore* ed inserire un prefisso fuorviante per un compilatore. Di conseguenza, con questo test possiamo comprendere l’importanza della fase di Data Cleaning e della funzione di *header_linkage()*.

Comune	Denominazione	Indirizzo
Mazara del Vallo	Museo del Satiro (Chiesa di Sant'Egidio)	Chiesa di Sant'Egidio, Piazza Plebiscito
Palazzolo Acreide	Museo regionale casa-museo Antonino Uccello a Palazzolo Acreide	Via Machiavelli, 19
Ispica	Area archeologica Parco Forza	Ispica, Via Cavagrande,snc
Gela	Castelluccio	Contrada Cucinella Spataro. SS 117 Bis
Terme Vigliatore	Villa Romana di San Biagio	Via Nazionale 3, localit� San Biagio
Gela	Antiquarium iconografico e Mura Timoleontee di Capo Soprano	Contrada Scavone - Viale Indipendenza
Monreale	Chiosstro Santa Maria la Nuova (Duomo)	Piazza Guglielmo il Buono
Noto	Area archeologica di Eloro	
Palermo	Palazzo Ajutamicristo	Via Garibaldi, 41
Siracusa	Percorso ipogeico di Piazza Duomo	Piazza Duomo 14
Mistretta	Museo delle Tradizioni silvo-pastorali di Mistretta	Via Libert� 184
Palermo	Real Albergo dei Poveri	Corso Calatafimi 217
Palermo	Villino Florio e Giardino	Via Regina Margherita
Palermo	Chiosstro di San Giovanni degli Eremiti	Via dei Benedettini 20
Villafranca Tirrena	Castello di Bauso	Piazza Castello
Piazza Armerina	Museo regionale della Villa Romana del Casale a Piazza Armerina	Piazza Armerina, Piazza Cattedrale 20 - Palazzo Trigona

Tabella 7: contenuto parziale del dataset musei_right

Scegliamo ora di usare la funzione *header_linkage()* e salvare i dataset ripuliti. A seguito di una ricerca empirica, abbiamo trovato come miglior valore di soglia per la funzione il valore “0.75”. Per *musei_right*, la fase di pulizia dei nomi attributo si limita a trasformare tutte le stringhe in minuscolo. Tuttavia, come mostrato in Tabella 8, per *musei_left*, il risultato della fase di Data Cleaning   determinante perch  ci permette di avere un significativo successo durante la fase di similarit  di stringhe per mezzo dell’algoritmo Jaro-Winker. Nello specifico, sono quattro i nomi di attributo individuati di cui la met , “*q1.2 sa indirizzo museo istituto*” – “*indirizzo*” e “*q1.8 sa telefono museo istituto*” – “*telefono*”, grazie alla pulizia dei dati.

Attributes found in header linkage are (First Dataset Attribute e Second Dataset Attribute):

['comune', 'provincia', 'q1.2 sa indirizzo museo istituto', 'q1.8 sa telefono museo istituto'] ['comune', 'provincia', 'indirizzo', 'telefono']

Musei_left	Data Cleaned Musei_left
Comune	comune
Provincia	provincia
Statale Ã†â€™?? Non statale	statale non statale
OMR	omr
Sistema museale territoriale	sistema museale territoriale
Sistema museale urbano	sistema museale urbano
Sistema museale tematico	sistema museale tematico
Casa museo	casa museo
Q1.1_SA_DENOMINAZIONE_MUSEO_ISTITUTO	q1.1 sa denominazione museo istituto
Q1.2_SA_INDIRIZZO_MUSEO_ISTITUTO	q1.2 sa indirizzo museo istituto
Q1.3_SA_LOCALITA_MUSEO_ISTITUTO	q1.3 sa localita museo istituto
Q1.4_SA_UBICAZIONE/EDIFICIO_MUSEO_ISTITUTO	q1.4 sa ubicazione/edificio museo istituto
Cap	cap
Q1.8_SA_TELEFONO_MUSEO_ISTITUTO	q1.8 sa telefono museo istituto
Q1.9_SA_FAX_MUSEO_ISTITUTO	q1.9 sa fax museo istituto
Q1.10_SA_EMAIL_MUSEO_ISTITUTO	q1.10 sa email museo istituto
Q1.11_SA_SITO_WEB_MUSEO_ISTITUTO	q1.11 sa sito web museo istituto
latitudine	latitudine
longitudine	longitudine

Tabella 8: Prima e dopo il Data Cleaning sui nomi degli attributi di musei_left

Per completare la ricerca delle associazioni sfruttando i valori attributo uguali presenti tra i due dataset, usiamo la funzione *record_linkage()* con tre scelte diverse rispetto ai valori di default: valori di soglia per entrambi i dataset pari a “0.1”, valore di soglia per l’algoritmo Jaro-Winkler pari a “0.98” e opzione *User* abilitata. La prima scelta è causata dalla presenza fuorviante di alcune parole con bassa occorrenza che andrebbero ad incidere sul risultato finale. Con il valore “0.1”

abbiamo trovato il limite inferiore minimo che risolvesse il problema di questi valori spuri e allo stesso tempo tenesse più largo possibile il range di valori considerati. La seconda scelta è stata resa necessaria perché alcuni *match* dell’algoritmo Jaro-Winkler non osservavano i requisiti necessari per essere considerati dei validi output. Di conseguenza, abbiamo ristretto il campo portandoci a un valore più alto rispetto allo standard “0.944”. Infine, la terza scelta scaturisce dalla presenza di alcuni errori ortografici in conflitto tra loro, come si può notare dall’output con sottostante con rispettive risposte dell’utente.

There are 3 Related Words. Insert the number to choose what word keep (1 or 2 or 0 to skip)

1 for: "museo della citta" o 2 for: "museo della citt"

1

Replaced "museo della citt" with: "museo della citta"

[...]

There are 1 Related Words. Insert the number to choose what word keep (1 or 2 or 0 to skip)

1 for: "dipende dal polo regionale di palermo per i parchi e i musei archeologici" o 2 for: "dipende dal polo regionale di palermo per i parchi e i musei archeologici."

Replaced "dipende dal polo regionale di palermo per i parchi e i musei archeologici" with: "dipende dal polo regionale di palermo per i parchi e i musei archeologici."

Abbiamo trovato, come si può notare nella Tabella 9, delle associazioni per mezzo delle parole: “museo”, “via” e “musei”, “piazza” e “archeologico”. In questo scenario i pesi sono intorno alla media dell’intervallo possibile tra zero e due. Tuttavia, come si può notare dall’output sottostante, i risultati sono molto soddisfacenti e rispecchiano le aspettative che ci siamo posti.

Attribute names found in record linkage are:

['q1.1 sa denominazione museo istituto', 'q1.2 sa indirizzo museo istituto', 'sistema museale urbano'] ['denominazione', 'indirizzo', 'categoria']

Infine, concateniamo i due dataset in un unico che salviamo in locale. Esso ha una colonna attributo aggiuntiva con i label “Lazio” e “Sicilia” per indicare da quale dataset provengono le informazioni.

Recommend concatenating from datasets, attributes:

{'comune': 'comune', 'provincia': 'provincia', 'q1.2 sa indirizzo museo istituto': 'indirizzo', 'q1.8 sa telefono museo istituto': 'telefono', 'q1.1 sa denominazione museo istituto': 'denominazione', 'sistema museale urbano': 'categoria'}

New dataset: musei_result.csv created.

Claim	Left Attribute Name	Right Attribute Name	Weight
museo	q1.1 sa denominazione museo istituto	denominazione	1,01
museo	q1.1 sa denominazione museo istituto	note	0,92
via	q1.2 sa indirizzo museo istituto	indirizzo	0,92
musei	sistema museale urbano	categoria	0,85
piazza	q1.2 sa indirizzo museo istituto	indirizzo	0,41
archeologo	q1.1 sa denominazione museo istituto	denominazione	0,23

Tabella 9: struttura decisionale di record_linkage() per il test Musei

8. Conclusioni e Sviluppi Futuri

Il progetto di tesi svolto è il primo passo di un più vasto studio nel campo degli Open Data, che ha lo scopo di fornire strumenti utili, originali e all'avanguardia che permettano agli utenti di interrogare i Dati Aperti con maggior profitto rispetto a quanto sia possibile fare attualmente.

L'obiettivo prefissato nel capitolo uno era di fornire agli utenti una libreria di metodi efficaci per ricercare, usare e confrontare le informazioni presenti sul portale dati.gov.it, individuando tra i dataset similarità che possano essere sfruttate per concatenare tra loro le informazioni.

I capitoli due, tre e quattro hanno il compito di mostrare i campi di ricerca utili per il progetto di tesi: gli Standard degli Open Data, la Data Fusione e il Record Linkage. Il loro studio e la comprensione dello stato dell'arte di ogni singolo settore ci ha permesso di poter ragionare sugli strumenti da implementare.

Lo stesso studio e impegno è stato necessario per la ricerca e l'utilizzo del migliore linguaggio di programmazione e relative librerie che fornissero il miglior piano di lavoro per il progetto da svolgere, come mostrato nel capitolo cinque. Nello specifico, Python si è dimostrato un linguaggio di programmazione estremamente utile e flessibile perché ha permesso di usare agevolmente tanto gli strumenti online, le Web API fornite dallo standard di metadata CKAN, quanto le librerie di Data Mining: Pandas e Jellyfish.

Nel capitolo sei, abbiamo spiegato le funzioni e gli algoritmi sviluppati, di cui è necessario approfondire i risultati raggiunti e riflettere su possibili idee per implementazioni future.

Il metodo di ricerca *search()* che abbiamo sviluppato è uno strumento di supporto di vitale importanza per l'intero progetto perché permette di lavorare su un unico programma senza bisogno di utilizzare l'interfaccia web via browser. Non essendo un componente centrale del progetto, la funzione di ricerca si limita a sfruttare le Web API fornite dal portale. Di conseguenza, uno sviluppo futuro potrebbe riguardare il miglioramento dello strumento di ricerca sfruttando tecniche più all'avanguardia che sfruttino le *regular expression* e correggano eventuali errori ortografici dell'utente come avviene nei moderni motori di ricerca.

Abbiamo dato particolari attenzioni alla funzione di estrazione *ext()* perché questo metodo può essere usato come base di partenza per molti altri progetti indipendentemente dal campo di ricerca che si voglia sfruttare. Nonostante sia uno strumento di supporto, qualsiasi lavoro che riguardi il portale dati.gov.it deve estrarre i dataset e fornirli in una forma manipolabile dall'utente. Inoltre, lo strumento implementato contiene delle variabili che se opportunamente modificate possono velocemente permetterne l'utilizzo anche su altri portali di Open Data. Ovviamente il metodo, che permette di aprire sotto forma di DataFrame i dataset in formato *csv*, *json*, *xml*, *xls*, *rdf*, *pdf* e *txt*, è stato implementato con l'idea di essere un utile strumento soprattutto per i file in formato *csv*. Infatti, tale formato è il protagonista indiscusso perché ben il 78% dei dataset forniti dal portale dati.gov.it lo usano. Per questo motivo, degli sviluppi futuri potrebbero riguardare una gestione più approfondita dei formati già disponibili, con particolare interesse per il formato *rdf* in quanto miglior formato standard per Open Data, e l'ampliamento del parco di formati utilizzabili aggiungendo dataset più specifici come quelli, per esempio, che contengono coordinate geospaziali.

Abbiamo visto durante l'esposizione di tutta la tesi che la funzione di Data Cleaning *preprocess()* è un tassello così importante del mosaico che non può essere considerato un semplice metodo di supporto. Essa è parte integrante del cuore del progetto in quanto ogni strumento più specifico implementato non avrebbe potuto esprimere le sue potenzialità senza un'importante fase di Data Cleaning. La libreria Record Linkage di Python è stata molto utile per questo obiettivo così come il controllo della presenza multipla dei caratteri di *spazio* alla fine delle stringhe che altrimenti renderebbe l'opzione *User* presente in molte funzioni completamente inutilizzabile in quanto ci sarebbero troppe parole ridondanti su cui scegliere. Inoltre, poiché lo scopo della tesi era fornire una solida base di partenza sul campo di ricerca degli Open Data, la concentrazione è stata massima per il tipo *stringa* a discapito di altri tipi, soprattutto quello dei valori numerici. Di conseguenza, uno sviluppo futuro potrebbe riguardare l'implementazione di nuovi metodi che consentano uno studio semantico sui valori numerici pari a quello fatto per le stringhe.

La funzione di ricerca di similarità tra i nomi degli attributi *header_linkage()* ha permesso di individuare un numero significativo di associazioni durante i test svolti. In particolare, è stata di estrema utilità per il test sulle anagrafiche dei musei intercettando un numero superiore di similarità rispetto alla funzione *record_linkage()*. L'utilizzo dell'algoritmo Jaro-Winkler e la sua raffinazione per mezzo di opportuni valori di soglia permette una grande usabilità dello strumento per molti dataset più o meno specifici. Da notare che il metodo si concentra sulla sintassi, ma in alcuni casi i nomi di attributo con nomi uguali potrebbe contenere valori semantici completamente diversi. Di conseguenza, un ulteriore sviluppo futuro potrebbe riguardare la similarità, e soprattutto la dissimilarità, del valore semantico dei nomi degli attributi.

La funzione di similarità tra i valori degli attributi tramite dizionari di occorrenze delle parole, *record_linkage()*, è il metodo che contiene l'algoritmo principale dell'intero progetto. Esso ha permesso di trovare associazioni tra attributi con nomi completamente diversi permettendo di automatizzare la concatenazione di dataset con risultati molto positivi. Il metodo, dopo un ampio Data Cleaning, sfrutta parole uguali presenti in attributi di diversi dataset per ipotizzare delle associazioni che l'utente potrà decidere se confermare. Così come la funzione *header_linkage()*, questo metodo controlla la sintassi delle parole. Di conseguenza, un ulteriore sviluppo futuro potrebbe riguardare la ricerca sul campo semantico, sfruttando in particolare il contesto dei sinonimi per una migliore omogeneità delle informazioni.

Il metodo di unione delle due funzioni visti in precedenza, *merge_keys()*, è uno strumento di supporto che automatizza la fusione dei risultati nel caso si volessero usare entrambi i metodi implementati di associazione per similarità. L'algoritmo concede maggiore priorità alla similarità tra i nomi attributo, ma non sempre è la scelta migliore. Di conseguenza, un ulteriore sviluppo futuro potrebbe riguardare l'aggiunta di ulteriori algoritmi di fusione in base alle esigenze.

Il settimo capitolo dei risultati sperimentali ha mostrato due scenari diversi in cui la libreria è stata messa alla prova.

Il primo test sui beni confiscati alla mafia ha mostrato l'utilità della libreria in un contesto in cui vi siano attributi con nomi diversi, ma che contengono un buon

quantitativo di parole uguali. L'utilità della funzione *record_linkage()* è stata elevata e le aspettative hanno ricevuto una risposta positiva grazie anche al supporto minore ma comunque importante della funzione *header_linkage()*.

Il secondo test sulle anagrafiche dei musei è un contesto più classico del Record Linkage, ma nel nostro caso i nomi e i valori degli attributi non erano sufficientemente comprensibili da un sistema informatico. Per questo motivo, abbiamo notato come una fase di Data Cleaning sia necessaria per un corretto funzionamento degli algoritmi sviluppati. La prova ha dato gran risalto alla funzione *header_linkage()* perché ha permesso di associare nomi di attributi chiaramente simili per un occhio umano, ma abbastanza complicati da valutare per un sistema informatico. Il metodo di *record_linkage()* ha permesso utili associazioni, ma i problemi di codifica e gli errori di ortografia di alcuni valori degli attributi hanno peggiorato le sue prestazioni. Infine, a differenza del test precedente che ha avuto un tempo di esecuzione estremamente veloce, questa prova ha necessitato circa quaranta secondi di elaborazione per la funzione di *record_linkage()* a causa dell'elevato numero di parole da ordinare all'interno dei dizionari. Di conseguenza, un ulteriore sviluppo futuro potrebbe riguardare la gestione e partizione delle risorse al fine di migliorare le prestazioni di elaborazione della libreria.

Bibliografia

- [1] «Wikipedia - Open government,» [Online]. Available: https://it.wikipedia.org/wiki/Open_government. [Consultato il giorno 28 02 2019].
- [2] F. P. D. R. Paolo Ciancarini, «Big Data Quality: a Roadmap for Open Data,» 2016.
- [3] «Open Government Partnership,» [Online]. Available: <https://www.opengovpartnership.org/about/about-ogp>. [Consultato il giorno 02 2019].
- [4] «Open Knowledge International,» [Online]. Available: <https://okfn.org/opendata/>. [Consultato il giorno 27 02 2019].
- [5] «Dati.Gov.it,» [Online]. Available: <https://www.dati.gov.it/>. [Consultato il giorno 28 02 2019].
- [6] «Python Documentation,» [Online]. Available: <https://docs.python.org/3/>. [Consultato il giorno 02 2019].
- [7] «Python Data Analysis Library,» [Online]. Available: <https://pandas.pydata.org/>. [Consultato il giorno 02 2019].
- [8] «Python Jellyfish,» [Online]. Available: <https://pypi.org/project/jellyfish/>. [Consultato il giorno 02 2019].
- [9] «Standard 25012,» [Online]. Available: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25012>. [Consultato il giorno 02 2019].
- [10] T. Berners-Lee, «Linked Data,» 27 07 2006. [Online]. Available: <https://www.w3.org/DesignIssues/LinkedData.html>. [Consultato il giorno 02 2019].
- [11] «Formato RDF,» [Online]. Available: <https://www.w3.org/RDF/>. [Consultato il giorno 02 2019].

- [12] «World Wide Web Consortium,» [Online]. Available: <https://www.w3.org/>. [Consultato il giorno 02 2019].
- [13] L. F. R. I. Antonio Corradi, «Linked Data for Open Government: the Case of Bologna».
- [14] «Wikipedia - Metadato,» [Online]. Available: <https://it.wikipedia.org/wiki/Metadato>. [Consultato il giorno 02 2019].
- [15] B. Lisowska, «Metadata for the open data portals,» 2016.
- [16] D. Laney, «3D Data Management: Controlling Data Volume, Velocity, and Variety,» *Application Delivery Strategies*, 2001.
- [17] Y. X. L. X. X. Z. Lili Zhang, «Multi-source Heterogeneous Data Fusion,» 2018.
- [18] X. N. Y. X. Wei Jiang, «Review on Big Data Fusion Methods of Quality Inspection for Consumer,» 2018.
- [19] A. Ruiz, «The 80/20 data science dilemma,» 26 9 2017. [Online]. Available: <https://www.infoworld.com/article/3228245/the-80-20-data-science-dilemma.html>. [Consultato il giorno 02 2019].
- [20] R. M. A. E.-G. e. al., «Record Linkage Approaches in Big Data: A State Of Art Study».
- [21] M. F. Halbert L. Dunn, «Record Linkage,» 1946.
- [22] A. B. S. Ivan P. Fellegi, «A Theory For Record Linkage».
- [23] M. A. Jaro, «Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida,» *Journal of the American Statistical Association*, 1989.
- [24] W. E. Winkler, «Overview of Record Linkage,» 2005.
- [25] «Wikipedia - Distanza di Hamming,» [Online]. Available: https://it.wikipedia.org/wiki/Distanza_di_Hamming. [Consultato il giorno 02 2019].

- [26] «Wikipedia - Levenshtein distance,» [Online]. Available: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance. [Consultato il giorno 02 2019].
- [27] «Wikipedia - Match rating approach,» [Online]. Available: https://en.wikipedia.org/wiki/Match_rating_approach. [Consultato il giorno 02 2019].
- [28] «Wikipedia - Jaro–Winkler distance,» [Online]. Available: https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance. [Consultato il giorno 02 2019].
- [29] «Kaggle Machine Learning & Data Science Survey 2017,» [Online]. Available: <https://www.kaggle.com/kaggle/kaggle-survey-2017>. [Consultato il giorno 02 2019].
- [30] «Python Record Linkage Toolkit Documentation,» [Online]. Available: <https://recordlinkage.readthedocs.io/>. [Consultato il giorno 02 2019].
- [31] «urllib — URL handling modules,» [Online]. Available: <https://docs.python.org/3/library/urllib.html>. [Consultato il giorno 02 2019].
- [32] «8.3. collections — High-performance container datatypes,» [Online]. Available: <https://docs.python.org/2/library/collections.html#ordereddict-objects>. [Consultato il giorno 02 2019].
- [33] «Dati.gov.it - Sviluppatori,» [Online]. Available: <https://www.dati.gov.it/content/sviluppatori>. [Consultato il giorno 02 2019].
- [34] «sociale-beni-confiscati-mafia,» [Online]. Available: <https://www.dati.gov.it/dataset/sociale-beni-confiscati-mafia>. [Consultato il giorno 01 2019].
- [35] «elenco-beni-confiscati-mafia,» [Online]. Available: https://www.dati.gov.it/sitesearch?search_api_views_fulltext=elenco-beni-confiscati-mafia. [Consultato il giorno 01 2019].

Appendice

In questa sezione della tesi, abbiamo inserito del materiale progettuale allo scopo di rendere più comprensibile il lavoro svolto. Nello specifico, abbiamo diviso questa sezione in tre parti. La prima parte contiene il codice scritto con nel linguaggio di programmazione Python di alcune funzioni rilevanti per il progetto di tesi. La seconda parte contiene i messaggi in output forniti dal programma durante i test del Capitolo 7. Infine, la terza parte contiene una porzione significativa dei dataset dallo stato originale fino allo stato finale.

A.1. Codice Sorgente

Abbiamo inserito in questa sezione dell'Appendice, il codice sorgente steso della funzione *ext()*, in Tabella 10, e della funzione *find_claims()*, in Tabella 11.

A.1.1. Funzione *ext()*

```
#Check file format and open it. url = package url, index = index column
#Format accepted: csv, json, excel, txt, rdf and pdf
def ext(url, index=None):
    #Manage file formats
    if 'csv' in url:
        print('Found file in format: CSV')
        count = 0
        #Try to open with ; separator
        try:
            if isinstance(index,int):
                df = pd.read_csv(url, sep=";", engine='python', index_col=index)
            else:
                df = pd.read_csv(url, sep=";", engine='python', index_col=None)
            if df.empty==True:
                dfExist=False
            else:
                dfExist = True
                sep = ";"
                count = count+1
        #Check on encode/decode and Parser errors
    except UnicodeDecodeError:
        print("Decode Error")
        if isinstance(index,int):
```

```

        df = pd.read_csv(url, sep=";", encoding='latin-1', engine='python',
index_col=index)
    else:
        df = pd.read_csv(url, sep=";", encoding='latin-1', engine='python',
index_col=None)
    except UnicodeEncodeError:
        print("Encode Error")
        if isinstance(index,int):
            df = pd.read_csv(url, sep=";", encoding='latin-1', engine='python',
index_col=index)
        else:
            df = pd.read_csv(url, sep=";", encoding='latin-1', engine='python',
index_col=None)
    except pd.errors.ParserError:
        print("Parser Error")
        dfExist = False

#Try to open with comma separator
try:
    if isinstance(index,int):
        dfcomma = pd.read_csv(url, sep=",", engine='python',
index_col=index)
    else:
        dfcomma = pd.read_csv(url, sep=",", engine='python',
index_col=None)
    if dfcomma.empty==True:
        dfcommaExist=False
    else:
        dfcommaExist = True
        sep = ","
        count = count+1
    except UnicodeDecodeError:
        print("Decode Error")
        if isinstance(index,int):
            dfcomma = pd.read_csv(url, sep=",", encoding='latin-1',
engine='python', index_col=index)
        else:
            dfcomma = pd.read_csv(url, sep=",", encoding='latin-1',
engine='python', index_col=None)
    except UnicodeEncodeError:
        print("Encode Error")
        if isinstance(index,int):
            dfcomma = pd.read_csv(url, sep=",", encoding='latin-1',
engine='python', index_col=index)
        else:
            dfcomma = pd.read_csv(url, sep=",", encoding='latin-1',
engine='python', index_col=None)
    except pd.errors.ParserError:

```

```

dfcommaExist = False

#Try to open with tab separator
try:
    if isinstance(index,int):
        dftab = pd.read_csv(url, sep="\t", engine='python', index_col=index)
    else:
        dftab = pd.read_csv(url, sep="\t", engine='python', index_col=None)
    if dftab.empty==True:
        dftabExist=False
    else:
        dftabExist = True
        sep="\t"
        count = count+1
except UnicodeDecodeError:
    print("Decode Error")
    if isinstance(index,int):
        dftab = pd.read_csv(url, sep="\t", encoding='latin-1', engine='python',
index_col=index)
    else:
        dftab = pd.read_csv(url, sep="\t", encoding='latin-1',engine='python',
index_col=None)
except UnicodeEncodeError:
    print("Encode Error")
    if isinstance(index,int):
        dftab = pd.read_csv(url, sep="\t", engine='python', index_col=index)
    else:
        dftab = pd.read_csv(url, sep="\t", engine='python', index_col=None)
except pd.errors.ParserError:
    dftabExist = False

#Try to open with whitespace separator
try:
    if isinstance(index,int):
        dfspace = pd.read_csv(url, sep=" ", engine='python',
index_col=index)
    else:
        dfspace = pd.read_csv(url, sep=" ", engine='python',
index_col=None)
    if dfspace.empty==True:
        dfspaceExist=False
    else:
        dfspaceExist = True
        sep = " "
        count = count+1
except UnicodeDecodeError:
    print("Decode Error")
    if isinstance(index,int):

```

```

        dfspace = pd.read_csv(url, sep=" ", encoding='latin-1',
engine='python', index_col=index)
    else:
        dfspace = pd.read_csv(url, sep=" ", encoding='latin-1',
engine='python', index_col=None)
    except UnicodeEncodeError:
        print("Encode Error")
        if isinstance(index,int):
            dfspace = pd.read_csv(url, sep=" ", encoding='latin-1',
engine='python', index_col=index)
        else:
            dfspace = pd.read_csv(url, sep=" ", encoding='latin-1',
engine='python', index_col=None)
    except pd.errors.ParserError:
        dfspaceExist = False

#Manage CSV separators error
if count == 0:
    sys.exit("Error: can't open this csv file. Exit")
elif count > 1:
    print("There are " + str(count) + " good separators.")
    if dfExist==True:
        print("Separator ;")
        pprint.pprint(df.head(1))
    if dfcommaExist==True :
        print("Separator ,")
        pprint.pprint(dfcomma.head(1))
    if dftabExist==True:
        print("Separator tab")
        pprint.pprint(dftab.head(1))
    if dfspaceExist==True :
        print("Separatore whitespace")
        pprint.pprint(dfspace.head(1))
    sep = input("Insert the chosen separator (; , tab space): ")

if sep==";":
    print("File opened correctly using the separator: ;")
    return df
elif sep==",":
    print("File opened correctly using the separator: ,")
    return dfcomma
elif sep=="tab":
    print("File opened correctly using the separator: tab")
    return dftab
elif sep==" ":
    print("File opened correctly using the separator: whitespace")
    return dfspace
else:

```



```

    print("The separator isn't valid")
elif 'json' in url:
    print('Found file in format: JSON')
    df = pd.read_json(url)
elif 'xml' in url:
    print('Found file in format: XML')
    if isinstance(index,int):
        df = pd.read_excel(url, encoding='utf-8',index_col=index)
    else:
        df = pd.read_excel(url, encoding='utf-8',index_col=None)
    return df
elif 'xls' in url:
    print('Found file in format: XLS')
    if isinstance(index,int):
        df = pd.read_excel(url, encoding='utf-8',index_col=index)
    else:
        df = pd.read_excel(url, encoding='utf-8',index_col=None)
    return df
elif 'rdf' in url:
    print('Found file in format: RDF')
    file = urllib.request.urlopen(url)
    g = rdflib.Graph()
    g.parse(file, format='xml')
    for stmt in g:
        pprint.pprint(stmt)
    return g
elif 'pdf' in url:
    print('Found file in format: PDF')
    df = read_pdf(url)
    return df
elif 'txt' in url:
    print('Found file in format: TXT')
    if isinstance(index,int):
        df = pd.read_csv(url, encoding='utf-8',index_col=index, sep = " ")
    else:
        df = pd.read_csv(url, encoding='utf-8',index_col=None, sep = " ")
else:
    sys.exit("This format isn't supported. Exit")

```

Tabella 10: codice sorgente della funzione ext()

A.1.2. Funzione `find_claims()`

```
#Find considerable words in columns of a dataset
#Parameters: DataFrame, Float (Occurrences threshold), String Array, Float,
True/False
def find_claims(df, threshold, stopwords, th_jaro, user):
    preprocess(df, th_jaro, user)
    count = 0
    dicts = [dict() for count in range(len(df.columns))]
    for col in df:
        #Split records in words/values using whitespaces as separators
        for val in df[col]:
            val = str(val).split(" ")
            #Count word occurrences and insert the result in a dictionary
            for word in val:
                if word in dicts[count]:
                    dicts[count][word] = dicts[count][word] + 1
                else:
                    dicts[count][word] = 1
            dicts[count] = OrderedDict(sorted(dicts[count].items(), key=lambda t: t[1],
reverse=True))
            count = count + 1

    indMerge= []
    dizMerge = []
    for col in dicts:
        ind = []
        diz = {}
        for key in col.keys():
            #Don't count null values
            if 'nan' not in key:
                if 'nan' in col.keys():
                    den = (df.shape[0] - col['nan'])
                    #Filter numbers, stopwords and values with an occurrences threshold
                    minor than threshold
                    if len(str(key)) > 1 and not(str(key).isdigit()) and
not(isfloat(str(key))) and str(key) not in stopwords and col[str(key)]/den >
threshold:
                        ind.append(key)
                        diz[str(key)] = col[str(key)]/den
                    else:
                        if len(str(key)) > 1 and not(str(key).isdigit()) and
not(isfloat(str(key))) and str(key) not in stopwords and col[str(key)]/df.shape[0]
> threshold:
                            ind.append(key)
                            diz[str(key)] = col[str(key)]/df.shape[0]
    indMerge.append(ind)
```

```
dizMerge.append(diz)
return indMerge, dizMerge
```

Tabella 11: codice sorgente della funzione *find_claims()*

A.2. Dataset

In questa sezione dell'Appendice, abbiamo inserito l'evoluzione dei dataset delle simulazioni: "Test su Beni Confiscati alla Mafia" e "Test su Anagrafica Museale".

A.2.1. Test su Beni Confiscati alla Mafia

In questa sezione abbiamo inserito: il dataset sorgente *mafia_left*, in Tabella 12 e la sua versione ripulita *new_mafia_left* in Tabella 13, il dataset sorgente *mafia_right* in Tabella 14 e la sua versione ripulita *new_mafia_right* in Tabella 15, e la concatenazione dei due dataset ripuliti *mafia_result* in **Errore. L'origine riferimento non è stata trovata.**

n. unità	tipologia	consistenza	ubicazione	concessionari	durata anni	progetto
1	abitazione	2,5 vani	Via Arquã	Fondazione Casa della Caritã Angelo Abriani Onlus Via Brambilla 8/1020128 Milano	10	Progetto Sociale
2	abitazione	3 vani	Via Arzaga	Fondazione Somaschi Onlus P.zza XXV Aprile 220121 Milano	6	Progetto Sociale
3	abitazione	3 vani	Via Arzaga	Fondazione Somaschi Onlus P.zza XXV Aprile 220121 Milano	6	Progetto Sociale
4	box	15 mq	Via Arzaga	Fondazione Somaschi Onlus P.zza XXV Aprile 220121 Milano	6	Progetto Sociale
5	box	15 mq	Via Asiago	Fondazione Somaschi Onlus	6	Progetto Sociale

				P.zza XXV Aprile 220121 Milano		
6	abitazione	2,5 vani	Via Baldinucci	Pio Istituto di Maternità PIM Via delle Camelie, 12 20147 Milano	5	Progetto Sociale
7	abitazione	5,5 vani	Via Baldo degli Ubaldi	Fondazione Casa della Carità Angelo Abriani Onlus Via Brambilla 8/1020128 Milano	in corso proroga	Progetto Sociale
8	abitazione	1,5 vani	Via Bassano del Grappa	Fondazione Progetto Arca Onlus Via S. Giovanni alla Paglia 720124 Milano	10	Progetto Sociale
9	abitazione	8 vani	Via Bembo	Fondazione Somaschi Onlus P.zza XXV Aprile 220121 Milano	6	Progetto Sociale
10	box	23 mq.	Via Bembo	Cooperativa Sociale Aromi a Tutto Campo Via Ippocrate, 4520161 Milano	3	Progetto Sociale
11	abitazione	1,5 vani	Via Bessarione	Gestione diretta Comune di Milano Settore Residenzialità		Progetto Sociale
12	abitazione	8 vani	Via Bianchi Mosca	Gestione diretta Comune di Milano Settore Residenzialità		Progetto Sociale

Tabella 12: porzione del dataset mafia_left

n. unità	tipologia	consistenza	ubicazione	concessionari	durata anni	progetto
1	abitazione	2,5 vani	via arqu	fondazione casa della carit angelo abriani onlus via bramilla 8/1020128 milano	10	progetto sociale
2	abitazione	3 vani	via arzaga	fondazione somaschi onlus p.zza xxv aprile 220121 milano	6	progetto sociale
3	abitazione	3 vani	via arzaga	fondazione somaschi onlus p.zza xxv aprile 220121 milano	6	progetto sociale
4	box	15 mq	via arzaga	fondazione somaschi onlus p.zza xxv aprile 220121 milano	6	progetto sociale
5	box	15 mq	via asiago	fondazione somaschi onlus p.zza xxv aprile 220121 milano	6	progetto sociale
6	abitazione	2,5 vani	via baldinucci	pio istituto di maternit pim via delle camelie, 12 20147 milano	5	progetto sociale

7	abitazione	5,5 vani	via baldo degli ubaldi	fondazione casa della carit angelo abriani onlus via brambilla 8/1020128 milano	in corso proroga	progetto sociale
8	abitazione	1,5 vani	via bassano del grappa	fondazione progetto arca onlus via s. giovanni alla paglia 720124 milano	10	progetto sociale
9	abitazione	8 vani	via bembo	fondazione somaschi onlus p.zza xxv aprile 220121 milano	6	progetto sociale
10	box	23 mq.	via bembo	cooperativa sociale aromi a tutto campovia ippocrate.4520161 milano	3	progetto sociale
11	abitazione	1,5 vani	via bessarione	gestione direttacomune di milano settore residenzialit		progetto sociale
12	abitazione	8 vani	via bianchi mos	gestione direttacomune di milano settore residenzialit		progetto sociale

Tabella 13: porzione del dataset new_mafia_left

numero	Immobilabile	Decreto Ministeriale	Data	Foglio	Particella	Sub	Tipologia	Associazione assegnataria	Note
10	Via Gambetta n 46	79223	03/12/1999	28	7870	30	villetta	Uff casa per alloggi di emergenza	nessuna
20	Villaggio cielo Azzurro - Baia dei Mori Vaccarizzo 4 strada "Via Gongola"	72941	30/11/1999	68	10910		villetta		Da ristrutturare
30	Via Nazario Sauro ang V D Chiesa	75519	30/11/1999	21	21020	100	appartamento		Riconsegnato sett 2013
40	Via Nazario	75519	30/11/1999	21	21020	10	appartamento	Uff casa per	nessuna

	Sauro ang V D Chiesa							alloggi di emergenza	
50	Via Nazario Sauro ang V D Chiesa	79221	03/12/1999	21	21020	80	appartamento	AVULSS - ONLUS	nessuna
60	Via Randazzo 27	79225	03/12/1999	13	7010		appartamento	ANVEZ	Non assegnabile ad usi civili H 2 40 Revoca assegnazione ott 2013
70	Via Randazzo 27	79225	03/12/1999	13	7160	10	giardino	ANVEZ	Non assegnabile ad usi civili H 2 40 Revoca assegnazione ott 2014
80	Via Masanello 4/a	75523	30/11/1999	28	20200	70	bottega	Ass Siculo-Rumena	Da ristrutturare
90	Via Anapo 16 — 18	76112	12/11/1999	17	1180	170	magazzino	Comune di Catania	nessuna
100	Via Anapo 16 — 18	76112	13/11/1999	17	1170	120	magazzino	CT FIADDA ONLUS	nessuna
110	Via Caprera 28/30	7521	18/11/1999	69/Z	14210		garage		demolito
120	Via Alfonzetti 73	49807	03/08/2000	69	161150	50	appartamento	Centro aiuto alla vita	nessuna

Tabella 14: porzione del dataset mafia_right

numero	immobili	decreto ministeriale	data	foglio	particella	sub	tipologia	associazione assegnataria	note
10	via gambetta n 46	79223	03/12/1999	28	7870	30	villetta	uff casa per alloggi di	nessuna

								emergenz a	
20	villaggi o cielo azzurro baia dei mori vaccariz zo 4 strada via gongola	72941	30/11/1 999	68	10910		villetta		da ristruttura re
30	via nazario sauro ang v d chiesa	75519	30/11/1 999	21	21020	100	appartam ento		riconsegn ato sett 2013
40	via nazario sauro ang v d chiesa	75519	30/11/1 999	21	21020	10	appartam ento	uff casa per alloggi di emergenz a	nessuna
50	via nazario sauro ang v d chiesa	79221	03/12/1 999	21	21020	80	appartam ento	avulss onlus	nessuna
60	via randazz o 27	79225	03/12/1 999	13	7010		appartam ento	anz	non assegnabi le ad usi civili h 2 40 revoca assegnazi one ott 2013
70	via randazz o 27	79225	03/12/1 999	13	7160	10	giardino	anz	non assegnabi le ad usi civili h 2 40 revoca assegnazi one ott 2014
80	via masanie llo 4/a	75523	30/11/1 999	28	20200	70	bottega	ass siculo rumena	da ristruttura re
90	via anapo 16 18	76112	12/11/1 999	17	1180	170	magazzin o	comune di catania	nessuna
100	via anapo 16 18	76112	13/11/1 999	17	1170	120	magazzin o	ct fiadda onlus	nessuna
110	via caprera 28/30	7521	18/11/1 999	69/z	14210		garage		demolito

120	via alfonzetti 73	49807	03/08/2 000	69	161150	50	appartam ento	centro aiuto alla vita	nessuna
-----	----------------------	-------	----------------	----	--------	----	------------------	------------------------------	---------

Tabella 15: porzione del dataset new_mafia_right

Column 1	_1	data	associazione assegnataria	immobile	note	tipologia
Milano	1520		angel service soc. coop. soc. arl onlusvia vallazze n. 10420131 milano	via vallazze	12 mq	cantina
Milano	1530		cooperativa sociale comunit del giambellinovia bellini gentile 620146 milano	via val isorno	4,5 vani	appart.
Milano	1540		fondazione exodus onlusvle marotta 18/2020134 milano	via varesina	105 mq	autosalon e e box di pertinenza
Milano	1550		fondazione exodus onlusvle marotta 18/2020134 milano	via varesina	11 mq	box
Milano	1560		associazione laltropallone ads onlusvia angera 320125 milano	via venini	21 mq	locale commerci ale
Catania	10	03/12/ 1999	uff casa per alloggi di emergenza	via gambetta n 46	nessuna	villetta
Catania	20	30/11/ 1999		villaggio cielo azzurro baia dei mori vaccarizzo 4 strada via gongola	da ristruttura re	villetta
Catania	30	30/11/ 1999		via nazario sauro ang v d chiesa	riconsegn ato sett 2013	appartame nto
Catania	40	30/11/ 1999	uff casa per alloggi di emergenza	via nazario sauro ang v d chiesa	nessuna	appartame nto

Tabella 16: porzione del dataset mafia_result

A.2.2. Test su Anagrafica Museale

In questa sezione abbiamo inserito: il dataset sorgente *musei_left*, in **Errore**. **L'origine riferimento non è stata trovata.** e la sua versione ripulita *new_musei_left* in Tabella 18, il dataset sorgente *musei_right* in Tabella 19 e la sua versione ripulita *new_musei_right* in Tabella 20, e la concatenazione dei due dataset ripuliti *musei_result* in Tabella 21 **Errore. L'origine riferimento non è stata trovata..**

Column1	Comune	Provincia	Q1.1_SA_DENOMINAZIONE_MUSEO_ISTITUTO	Q1.2_SA_INDIRIZZO_MUSEO_ISTITUTO	Q1.3_SA_LOCALITA_MUSEO_ISTITUTO	Q1.8_SA_TELEFONO_MUSEO_ISTITUTO
0	Farnese	Viterbo	museo civico FERRANTE RITTATORE VONWILLER	VIA COLLE SAN MARTINO; 16	Farnese	761458849
1	Roma	Roma	ECOMUSEO DEL LITORALE ROMANO	VIA DEL FOSSO DI DRAGONCELLINO; 172	LONGARINA/OSTIA	65651764
2	Colleparado	Frosinone	Abbazia di Trisulti	Via Trisulti; 8	-	77547024
3	Roma	Roma	MUSEO STORICO DELLA MOTORIZZAZIONE MILITARE DELL'ESERCITO	VIALE DELL'ESERCITO; 170	CECCHIGNOLA	06 50237374
4	Roma	Roma	MUSEO STORICO DELLA COMUNICAZIONE	VIALE EUROPA 190	-	06 54443000
5	Roma	Roma	Complesso di Capo di Bove	Via Appia Antica; 222	-	67806686
6	Roma	Roma	MUSEI CAPITOLINI	PIAZZA DEL CAMPIDOGGIO; 1		667102475
7	Roma	Roma	MUSEO TEATRALE SIAE DEL BURCARDO	VIA DEL SUDARIO; 44	-	66819471
8	Roma	Roma	MUSEO CRIMINOLOGICO	VIA DEL GONFALONE; 29	-	668899442
9	Roma	Roma	Museo d'arte sacra di San Giovanni dei Fiorentini	via degli Acciaioli 2		06 6869892

Tabella 17: porzione del dataset musei_left

Column1	comune	provincia	q1.1 sa denominazione museo istituto	q1.2 sa indirizzo museo istituto	q1.3 sa localita museo istituto	q1.8 sa telefono museo istituto
0	farnese	viterbo	museo civico ferrante rittatore vonwiller	via colle san martino 16	farnese	761458849
1	roma	roma	ecomuseo del litorale romano	via del fosso di dragoncello 172	longarina/ostia	65651764
2	colleparado	frosinone	abbazia di trisulti	via trisulti 8		77547024
3	roma	roma	museo storico della motorizzazione militare dellesercito	viale dellesercito 170	cecchignola	06 50237374
4	roma	roma	museo storico della comunicazione	viale europa 190		06 54443000
5	roma	roma	complesso di capo di bove	via appia antica 222		67806686
6	roma	roma	musei capitolini	piazza del campidoglio 1		667102475
7	roma	roma	museo teatrale siae del burcardo	via del sudario 44		66819471
8	roma	roma	museo criminologico	via del gonfalone 29		668899442
9	roma	roma	museo darte sacra di san giovanni dei fiorentini	via degli acciaioli 2		06 6869892

Tabella 18: porzione del dataset new_musei_left

Column1	Categoria	Provincia	Comune	Denominazione	Indirizzo	Telefono
0	Musei archeologici	Trapani	Mazara del Vallo	Museo del Satiro (Chiesa di Sant'Egidio)	Chiesa di Sant'Egidio, Piazza Plebiscito	0923933917

1	Musei etnoantropologici	Siracusa	Palazzolo Acreide	Museo regionale casa-museo Antonino Uccello a Palazzolo Acreide	Via Machiavelli, 19	0931 881499
2	Aree archeologiche	Ragusa	Ispica	Area archeologica Parco Forza	Ispica, Via Cavagrande, snc	0932952 608
3	Aree archeologiche	Caltanissetta	Gela	Castelluccio	Contrada Cucinella Spataro. SS 117 Bis	0933912 626
4	Aree archeologiche	Messina	Terme Vigliatore	Villa Romana di San Biagio	Via Nazionale 3, localit� San Biagio	0909740 488

Tabella 19: porzione del dataset musei_right

Column1	categoria	provincia	comune	denominazione	indirizzo	telefono
0	musei archeologici	trapani	mazara del vallo	museo del satiro chiesa di santegidio	chiesa di santegidio, piazza plebiscito	0923933 917
1	musei etnoantropologici	siracusa	palazzolo acreide	museo regionale casa museo antonino uccello a palazzolo acreide	via machiavelli, 19	0931 881499
2	aree archeologiche	ragusa	ispica	area archeologica a parco forza	ispica, via cavagrande, snc	0932952 608
3	aree archeologiche	caltanissetta	gela	castelluccio	contrada cucinella spataro. ss 117 bis	0933912 626
4	aree archeologiche	messina	terme vigliatore	villa romana di san biagio	via nazionale 3, localit san biagio	0909740 488

Tabella 20: porzione del dataset new_musei_right

Column 1	_1	comune	denominazione	indirizzo	provincia	telefono
Lazio	352	tuscania	raccolta di montebello del maestro giuseppe cesetti	centro strada montebello 3	viterbo	761442695
Lazio	353	tuscania	museo archeologico nazionale	largo mario moretti 1	viterbo	0761 436209
Lazio	354	vallepietra	sale museali ex voto santuario santissima trinita valle pietra	santuario santissima trinita	roma	774899084
Lazio	355	viterbo	confraternita ss.sacramento e s.rosario	piazza oratorio 5	viterbo	761379301
Lazio	356	viterbo	area archeologica antica citt di ferento	strada provinciale tenerina km 8000	viterbo	
Sicilia	0	mazara del vallo	museo del satiro chiesa di santegidio	chiesa di santegidio, piazza plebiscito	trapani	0923933917
Sicilia	1	palazzolo acreide	museo regionale casa museo antonino uccello a palazzolo acreide	via machiavelli, 19	siracusa	0931 881499
Sicilia	2	ispica	area archeologica parco forza	ispica, via cavagrande, snc	ragusa	0932952608
Sicilia	3	gela	castelluccio	contrada cucinella spataro. ss 117 bis	caltanissetta	0933912626
Sicilia	4	terme vigliatore	villa romana di san biagio	via nazionale 3, localit san biagio	messina	0909740488

Tabella 21: porzione del dataset musei_result

Ringraziamenti

Dopo sei mesi di lavoro prettamente formale, trovo doveroso dedicare qualche minuto per uscire dagli schemi accademici e dedicare il meritato spazio a chi ha reso tutto questo possibile.

Voglio ringraziare per primo il mio relatore, il Prof. Claudio Sartori, che ha sempre accettato le mie opinioni e le ha rese propositive grazie alla sua esperienza e abilità, non solo tecnica, ma soprattutto sociale. Questa tesi nasce da una sua idea ed è stata plasmata dalle sue direttive unite alla passione che mi ha indotto con ragionamenti mai banali.

Questo obiettivo non sarebbe mai stato possibile da raggiungere senza due persone, Diego e Giovanna, i miei genitori, che mi hanno sempre donato tutto l'amore che hanno in corpo. Voglio ringraziarli perché mi hanno insegnato tutto quello che serve per completare un percorso di studi e, soprattutto, per vivere. Due persone diverse nei loro modi, ma da cui ho imparato tanto a pazientare quanto ad agire. Il mio orgoglio per come affrontano ogni giorno la vita non è inferiore al loro per l'obiettivo oggi conseguito.

Il mio percorso di studi non è stato lineare. Le difficoltà economiche non mi hanno permesso di iniziare subito gli studi magistrali, ma la decisione di intraprendere di nuovo l'esperienza accademica è frutto di dedizione e ragionamenti che ho condiviso con una persona che amo come me stesso. Voglio ringraziare la mia fidanzata Maria Elena per avermi supportato, e sopportato, ogni giorno durante l'intero percorso che ci ha portati qui.

Lo studio presso l'Università di Bologna mi ha insegnato che ogni scopo deve essere raggiunto un pezzo alla volta. Questo ragionamento mi ha permesso di capire quanto io sia riconoscente nei confronti dei miei amici. Voglio ringraziare ognuno di loro per avermi dato la forza nei momenti di difficoltà e fatto tornare con i piedi per terra quando vi era un'immotivata esaltazione. Non citerò i loro nomi perché, come accennato in precedenza, un mosaico è formato di tanti piccoli tasselli che formano un'opera inestimabile. Anche se ci sono tessere che risaltano più all'occhio di altre, il suo valore diminuisce egualmente se un tassello, indipendentemente dalla sua posizione, manca.

Un ringraziamento speciale va a mio cugino Alessandro per aver instillato in me la passione per la scienza dell'informazione sin da piccolo, avviando un processo di amore irreversibile per la materia. Infine, voglio ringraziare i miei parenti che nel momento del bisogno non si sono mai tirati indietro nel darmi una mano, tanto quelli di sangue e non di meno quelli acquisiti.