UNIVERSITÀ DEGLI STUDI

DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,

Informatiche e Matematiche

Corso di Laurea in Informatica

Data analytics su una sperimentazione clinica per healthy aging

Francesco Ghinelli

Tesi di Laurea

Relatori:

Prof. Riccardo Martoglia

Prof. Federica Mandreoli

Sommario

Sommario figure

Sommario script

Introduzione

Nell' epoca attuale, nella quale l'innovazione tecnologica porta migliorie e supporti alla vita di tutti i giorni, anche in ambito medico sono sempre più presenti i sistemi informatici come smartphone e smartwatch che ci seguono e ci monitorano nelle azioni quotidiane. Proprio su questi aspetti si basa lo studio MySAwH, infatti, questo studio, condotto su un discreto numero di pazienti anziani affetti da HIV, vuole tenere sotto controllo lo stato mentale e fisico degli stessi. Per monitorare i pazienti, oltre alle comuni visite mediche, sono stati creati dei questionari volti a comprendere il loro stato di salute psicologica, proposti agli stessi pazienti mediante l'ausilio di una apposita applicazione per dispositivi smartphone. Inoltre, per registrare l'attività fisica dei pazienti, sono stati forniti loro dei dispositivi smartwatch Garmin.

Il presente elaborato vuole descrivere il mio ruolo e le operazioni da me effettuate nell'ambito dello studio MySAwH. Questo studio, condotto dal professore Giovanni Guaraldi, vuole individuare le correlazioni presenti tra diversi punteggi associati a pazienti anziani affetti da HIV, così da poterne individuare e prevedere lo stato di salute fisica e psicologica.

Nell'ambito di questo studio si usano indici già conosciuti in ambito medico (FI, KATZ, IADL, EQ5D5L) e un nuovo indice (IC), il quale valuta le capacità psico-motorie dei pazienti, del quale si vuole individuare l'efficacia nel prevedere lo stato mentale dei pazienti.

Questo elaborato è articolato in sette capitoli: nel primo capitolo vado a descrivere alcuni termini importanti per questa tesi, il mio ruolo nel progetto MySAwH e i passi fondamentali da compiere in un processo di data analisi. Nel secondo capitolo mi sono concentrato sugli strumenti da me utilizzati per effettuare le operazioni sui dati e per visualizzarne i risultati. Nel terzo capitolo approfondisco tutte le caratteristiche dello studio MySAwH e gli obbiettivi che si è posto. Nel quarto capitolo vado ad analizzare le operazioni da me effettuate per ripulire e riorganizzare i dati in modo utile allo studio. Nel quinto capitolo analizzo nel dettaglio i dati, descrivendo le modalità di

rappresentazione che si sono rese necessarie per poter aver una idea chiara degli stessi, della loro quantità e dei diversi problemi che li affliggono. Nel sesto capitolo vado a calcolare gli indici e a descrivere le modalità per la rappresentazione degli indici con le quali si è potuto effettivamente osservare lo stato di salute dei pazienti. Nel settimo e ultimo capitolo mi sono concentrato sulle correlazioni tra i diversi dati necessarie per capire come i diversi indici si rapportano l'uno con l'altro e se l'indice di capacità intrinseca correla in modo utile con gli altri indici.

Parte I: Caso di studio

Capitolo 1: Data science

In questo capitolo vado a descrivere alcune dei termini importanti per questa tesi e il processo di analisi dei dati proprio della data science.

1.1 Introduzione alla data science

Per poter parlare di data science bisognerà prima capire cosa è un dato.

In informatica, il termine "dato"[1] indica un valore, tipicamente numerico in bit, che può essere elaborato e/o trasformato da un automa o meglio da un elaboratore elettronico. Il dato rappresenta l'oggetto specifico su cui interviene l'esecutore dell'algoritmo. Le memorie di massa consentono di salvare i dati in modo permanente, mentre il processo di registrazione dei dati in una memoria si chiama memorizzazione o archiviazione. Singoli dati o più spesso insiemi di dati possono essere conservati in file o in database.

Tendenzialmente un dato è un segnale digitale associato mediante un determinato codice ad un segnale o ad un sistema di simboli come il codice ASCII, che permette di rappresentare in binario numeri, lettere dell'alfabeto e simboli.

Detto ciò, possiamo accingerci a specificare il significato di data science.

La data science[2], in italiano scienza dei dati, è un insieme di principi metodologici e tecniche multidisciplinari volte a interpretare ed estrarre conoscenza dai dati attraverso la relativa fase di analisi da parte di un esperto (*Data Scientist*). I metodi della scienza dei dati (spesso associati al concetto di data mining) si basano su tecniche provenienti

da varie discipline, principalmente da matematica, statistica, scienza dell'informazione, informatica e scienze sociali, in particolar modo nei seguenti sottodomini: basi di dati e visualizzazione dati o business intelligence, intelligenza artificiale o apprendimento automatico.

Uno dei fattori che ha portato successo a questa scienza è stato l'avvento dei Big Data e l'idea del "valore del dato", propria di questo paradigma, è evoluto il concetto di Scienza dei dati, il cui principio fondante non è la mera gestione del dato, ma una più ampia valorizzazione della grande mole eterogenea di dati proveniente da diverse fonti (data warehouse, sensori, web, ecc..).

La scienza dei dati al giorno d'oggi va quindi intesa come una disciplina trasversale, cui fanno capo sia le sfere dell'informatica, della statistica e della matematica, come nell'accezione originale, sia un insieme di competenze più manageriali, legate alla più recente necessità di sapere leggere, interpretare e capitalizzare i dati a fini di business.

Questa scienza può essere quindi applicata a tutti i possibili ambiti: da quelli economici come possono essere le banche (che necessitano di archiviare dati dei clienti), supermercati (che necessitano di archiviare i dati dei prodotti in vendita); a quelli medici, come gli ospedali (che necessitano di archiviare dati riguardanti i pazienti), ambito che verrà analizzato nel dettaglio nella seconda parte di questa tesi.

1.2 Big Data

Nel paragrafo precedente abbiamo accennato il concetto di Big Data, terminologia nata negli anni 90 a seguito dell'aumento in modo spropositato dei dati prodotti dalle aziende.

Questo concetto viene descritto come: una raccolta di dati così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per

l'estrazione di valore o conoscenza[3]. Questo concetto è utilizzato in riferimento alla capacità (propria della scienza dei dati) di analizzare ovvero estrapolare e mettere in relazione un'enorme mole di dati eterogenei, strutturati e non strutturati, con lo scopo di individuare legami tra fenomeni diversi (ad esempio correlazioni) e prevedere quelli futuri. Con i big data la mole dei dati è dell'ordine degli zettabyte, ovvero miliardi di terabyte. Si parla quindi di big data quando si ha un insieme talmente grande e complesso di dati che richiede la definizione di nuovi strumenti e metodologie per estrapolare, gestire e processare informazioni entro un tempo ragionevole[4]. Non esiste una dimensione di riferimento, ma questa cambia sempre, poiché le macchine sono sempre più veloci e i data set sono sempre più grandi. Secondo uno studio del 2001, l'analista Douglas Laney aveva definito il modello di crescita come tridimensionale (modello delle "3V"[5]): con il passare del tempo aumentano volume (dei dati), velocità e varietà (dei dati). In molti casi questo modello è ancora valido, nonostante esso sia stato successivamente esteso. Il primo modello di Douglas Laney, il modello delle "3V", comprende:

Volume: si riferisce alla quantità di dati (strutturati, non strutturati) generati ogni secondo. Tali dati sono generati da sorgenti eterogenee quali: sensori, log, eventi, e-mail, social media e database tradizionali;

Varietà: si riferisce alla differente tipologia dei dati che vengono generati, collezionati ed utilizzati. Prima dell'epoca dei big data si tendeva a prendere in considerazione per le analisi principalmente dati strutturati e la loro manipolazione veniva eseguita mediante uso di database relazionali. Per avere analisi più accurate e più profonde, oggi è necessario prendere in considerazione anche dati non strutturati (ad esempio file di testo generati dalle macchine industriali o i log di web server o di firewall) e semi strutturati (ad esempio atto notarile con frasi fisse e frasi variabili) oltre che quelli strutturati (ad esempio tabella di un database);

Velocità: si riferisce alla velocità con cui i nuovi dati vengono generati. Non solo la celerità nella generazione dei dati, ma anche la necessità che questi dati/informazioni arrivino real-time al fine di effettuare analisi su di essi.

Con il tempo, sono state introdotte una quarta V, quella di veridicità, e poi una quinta, quella di valore[6].

Veridicità: considerando la varietà dei dati sorgente (dati strutturati o non strutturati) e la velocità alla quale tali dati possono variare, è molto probabile che non si riesca a garantire la stessa qualità di dati in ingresso ai sistemi di analisi normalmente disponibile in processi di ETL tradizionali. È evidente che se i dati alla base delle analisi sono poco accurati, i risultati delle analisi non saranno migliori. Visto che su tali risultati possono essere basate delle decisioni, è fondamentale assegnare un indice di veridicità ai dati su cui si basano le analisi, in modo da avere una misura dell'affidabilità;

Valore: si riferisce alla capacità di trasformare i dati in valore. Un progetto big data necessita di investimenti, anche importanti, per la raccolta granulare dei dati e la loro analisi. Prima di avviare un'iniziativa è importante valutare e documentare quale sia il valore effettivo portato al business.

1.3 Come opera la scienza dei dati

Siccome la scienza dei dati abbraccia la maggior parte di ciò che ci circonda, spesso ciò che vogliamo rappresentare appartiene a un insieme di dati non numerici e non strutturati, dove strutturato indica una rappresentazione dei dati ordinata per righe e colonne. Proprio per questo motivo i dati dovranno essere analizzati e in un modo o in un altro inseriti all'interno di un database, così da renderli facilmente e soprattutto velocemente reperibili. Per rendere facilmente reperibile un dato in un database spesso si usano delle chiavi, ossia degli identificatori univoci associati agli oggetti.

Un esempio di questo tipo può essere un paziente di un ospedale, se volessimo trovare questo paziente in un database dovremmo inserire tutte le sue caratteristiche che lo distinguono dagli altri (nome, cognome, data di nascita, luogo di nascita, …), questa

operazione risulta molto lenta, per questo motivo sono stati creati i codici fiscali che consentono di individuare rapidamente un paziente presente nei database degli ospedali.

Ma tutto questo non viene effettuato da un analista informatico (data scientist), non è lui che crea i codici fiscali, le operazioni effettuate da un analista sono riassunte nelle cosiddette "4 A": Architettura, Acquisizione, Analisi e Archiviazione[7].

Architettura: sta alla base di tutto, infatti l'analista dovrà essere presente nelle fasi di definizione del progetto di un software per definire al meglio il formato dei dati e la loro organizzazione, in modo che questi siano utili nelle successive fasi di analisi, presentazione e visualizzazione;

Acquisizione: è una fase molto delicata di questo processo, infatti tutte le volte che un utente ha il controllo di un sistema o di una sua parte bisogna assicurarsi che le operazioni da lui eseguite siano eseguite rispettando certi canoni: unità di misura, formato dei dati inseriti, formato dei dati assenti, formato delle date. Inoltre, il formato di dato ottenuto dalla acquisizione dovrà rispettare le scelte fatte in fase di sviluppo della architettura;

Analisi: è la fase in cui il lavoro dell'analista è più essenziale. In questa fase l'analista si occupa di prendere i dati raccolti estrarne un significato, rappresentare i dati in un formato comprensibile attraverso tabelle e grafici. Questa fase racchiude aspetti tecnici, matematici e statistici ed è necessaria per far sì che il committente e/o le persone che faranno uso di questi dati possano farlo in modo semplice e intuitivo. Infatti, analisi scientifiche complesse sono inutili se non possono essere rappresentate a colui che le ha richieste in un modo a lui comprensibile;

Archiviazione: è la fase finale di questo processo, per quanto possa sembrare una fase semplice, anch'essa presenta delle complicazioni, infatti lo scopo principale di questa fase è quello di archiviare i dati in caso di riusi successivi, ma spesso risulta complesso anticipare una possibile modalità di utilizzo di questi dati.

Oltre a queste fasi tecniche un analista dovrà:

- Conoscere il dominio dell'applicazione che andrà a creare;
- Comunicare con gli utenti e con gli esperti del dominio;
- Sviluppare una visione d'insieme del prodotto che andrà a creare;
- Individuare il modo migliore per rappresentare i dati;
- Saper trasformare e analizzare i dati;
- Avere attenzione alla qualità, anche se il data set iniziale non è buono;
- Avere rispetto della privacy.

1.3.1 Più nel dettaglio

Nel dettaglio si possono individuare cinque diverse fasi della data science (Figura 1-1 Fasi della data science), anche se mi concentrerò principalmente sulle prime tre.

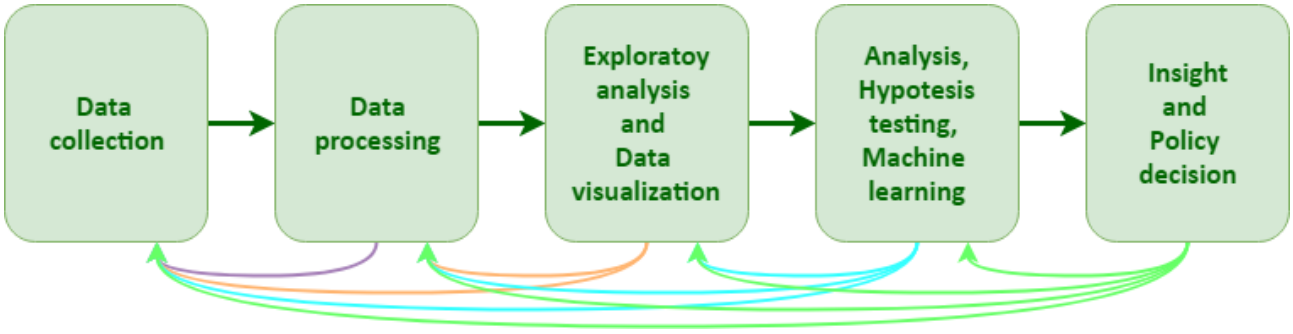


Figura 1-1 Fasi della data science

Queste fasi si susseguono seguendo il flusso descritto dalle frecce scure, ma spesso arrivati a una fase si possono individuare problemi che riportano lo studio a una delle fasi precedenti.

1.3.2 Data collection

La data collection (raccolta dati) è l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di informazioni utili da grandi quantità di dati (es. database, datawarehouse ecc..), attraverso metodi automatici o semi-automatici

(es. machine learning) e l'utilizzo scientifico, aziendale/industriale o operativo delle stesse.

Questi dati devono essere raccolti dall'analista oppure vengono forniti dal committente del software. Spesso si tende a utilizzare dati già in un formato comprensibile a una macchina come ad esempio tabelle Excel, database SQL o file csv.

Come già detto, uno degli aspetti principali di cui si deve occupare la data science è proprio la qualità di questi dati [8], la quale può essere compromessa da:

Rumore: si tratta della distorsione dei dati, questa distorsione necessita di essere rimossa o comunque ridotta prima che gli algoritmi di data mining vengano applicati ai dati.

Valori anomali: (in inglese outlier) sono valori evidentemente diversi rispetto a quelli presenti nel data set. Questi valori possono essere legati a caratteristiche effettivamente utili oppure dovuti a inserimenti sbagliati, in entrambi i casi possono incidere su eventuali analisi statistiche effettuate sui dati, risultando per tanto problematici.

Valori mancanti: corrispondono a caratteristiche di oggetti mancanti nel database. Per risolvere questo problema possiamo: rimuovere l'istanza degli oggetti con valori mancanti, stimare i valori mancanti oppure ignorare i valori mancanti quando eseguiamo gli algoritmi di data mining.

Duplicati: si presentano quando diverse istanze di stessi oggetti hanno le stesse caratteristiche, queste possono essere rimosse o tenute basandosi sul contesto.

1.3.3 Data processing

Spesso i dati forniti non sono pronti per essere direttamente utilizzati, per questo motivo si necessita di una fase di processo dei dati[9] per effettuare operazioni di:

Validazione: processo atto a garantire che i dati siano puliti e di qualità. La convalida dei dati ha lo scopo di fornire garanzie ben definite per quanto riguarda l'idoneità,

l’accuratezza e l’uniformità per qualsiasi tipo di input da parte di un utente di una applicazione o in un sistema automatizzato;

Ordinamento: consente di ordinare i dati in sequenze o categorie ben definite;

Aggregazione: si occupa di combinare caratteristiche multiple in una singola più complessa oppure di aggiungere caratteristiche a oggetti;

Discretizzazione: consente di convertire valori continui in valori discreti, utilizzando dei range per applicare le conversioni;

Selezione delle caratteristiche: spesso tutte le caratteristiche non sono necessarie, in tal caso si puo andare a selezionare un sottoinsieme di caratteristiche in modo da migliorare le prestazioni dell’algoritmo di data mining;

Estrazione delle caratteristiche: consente la conversione dell’attuale set di caratteristiche in un nuovo set che possa effettuare i task di data mining in modo migliore;

Campionatura: spesso processare l’intero insieme dei dati è un’operazione costosa in termini di tempo, per motivo la campionatura consente di estrarre piccoli subset di dati da usare per le operazioni di data mining. Una corretta scelta dei campioni consente di rappresentare in modo corretto il dominio dei dati, assicurando che i risultati ottenuti con i campioni sono vicini a quelli ottenuti con l’intero data set.

La fase di data processing prende in ingresso uno o più data set e produce uno o più data set applicando le opportune modifiche ai dati.

1.3.4 Exploratory analysis and data visualization

Nell'ambito della scienza dei dati, l'analisi esplorativa dei dati è un processo di ispezione, pulizia, trasformazione e modellazione dei dati con il fine di evidenziare informazioni che suggeriscano conclusioni e supportino le decisioni strategiche aziendali. L'analisi di dati ha molti approcci e sfaccettature, il che comprende

tecniche diversissime tra loro che si riconoscono con una serie di definizioni varie nel commercio, le scienze naturali e sociali.

Il data mining è una tecnica particolare di analisi dei dati che si focalizza nella modellazione e scoperta di conoscenza per scopi predittivi piuttosto che descrittivi. La business intelligence identifica l'analisi di dati che si basa fondamentalmente sull'aggregazione, focalizzandosi sulle informazioni aziendali. Nell'ambito dei big data si parla di big data analytics. Nelle applicazioni statistiche, gli studiosi dividono l'analisi dei dati in statistica descrittiva, analisi dei dati esplorativa (ADE)[10] e analisi dei dati di conferma (ADC). L'ADE si concentra sullo scoprire nuove caratteristiche presenti nei dati, mentre l'ADC nel confermare o falsificare le ipotesi esistenti. L'analisi predittiva si concentra sull'applicazione di modelli statistici o strutturali per classificazione o il forecasting predittivo, mentre l'analisi testuale applica tecniche statistiche, linguistiche e strutturali per estrarre e classificare informazioni da fonti testuali, una categoria di dati non-strutturati.

La visualizzazione dei dati è vista da molte discipline come un equivalente moderno della comunicazione visiva. Per comunicare informazioni in modo efficiente, la visualizzazione dei dati usa grafici statistici, plot, grafici informativi e altri strumenti. I dati numerici possono essere codificati usando punti, linee, barre, colori, per comunicare visivamente un messaggio quantitativo[11]. La visualizzazione è molto utile agli utenti, consente infatti di rendere i dati più accessibili e comprensibili, per questo motivo i grafici devono:

- Visualizzare i dati;
- Indurre colui che li guarda alla sostanza piuttosto che alla metodologia, al design, alla tecnologia usata;
- Evitare di distorcere il significato dei dati;
- Rendere grandi data set coerenti;
- Consentire di comparare differenti dati;
- Consentire la visualizzazione dei diversi livelli dei dati;
- Essere integrati con descrizioni statistiche e verbali del data set.

Capitolo 2: Strumenti

In questo capitolo vado a descrivere gli strumenti software da me utilizzati per le operazioni sui dati, che si distinguono tra: linguaggi di programmazione, librerie scientifiche per le operazioni di analisi e correlazione dei dati e librerie grafiche per la visualizzazione dei risultati.

2.1 Linguaggio di programmazione

Il linguaggio di programmazione che è stato scelto per la creazione dei diversi programmi di estrazione, pulizia e visualizzazione dei dati è Python nella versione 3, la scelta di un linguaggio dinamico e flessibile come Python ha consentito di effettuare in modo semplice operazioni molto complesse che altrimenti avrebbero richiesto più tempo e più codice per essere realizzate.

2.2 NumPy

È un pacchetto fondamentale per calcoli scientifici in Python. NumPy[12] è una libreria Python che fornisce array multidimensionali, oggetti derivati da essi (maschere e matrici) e un set di routine per effettuare operazioni veloci sugli array, incluse operazioni matematiche, logiche, algebriche, statistiche, di manipolazione, di ordinamento, di selezione, di I/O e tante altre.

Alla base del pacchetto NumPy ci sono gli ndarray, questi oggetti sono in grado di incapsulare array n-dimensionali, contenenti tipi di dato omogenei e di effettuare delle operazioni sugli stessi.

Ci sono molte differenze tra gli array NumPy e gli array standard di Python:

Dimensione fissata al momento della creazione: La dimensione degli array NumPy è fissata al momento della creazione, al contrario delle liste Python che possono crescere dinamicamente. Cambiare dimensione a un array NumPy ne comporta la cancellazione e la creazione di uno nuovo;

Tipizzazione statica: Gli elementi degli array NumPy sono tutti dello stesso tipo di dato;

Facilitazione delle operazioni matematiche: Gli array NumPy facilitano le operazioni matematiche e di altri tipi su grosse quantità di dati, riducendo l'uso di codice rispetto a quello che verrebbe usato per effettuare le stesse operazioni in Python.

Un insieme crescente di pacchetti scientifici e matematici Python fa uso degli array NumPy, convertendo oggetti Python in array NumPy prima di procedere a effettuare le operazioni matematiche su di essi, per poi ritornare i risultati.

2.3 SciPy

SciPy[13] è una libreria contenente una collezione di algoritmi e funzioni, realizzate sulla libreria NumPy. Questa libreria fornisce comandi di alto livello e classi per la manipolazione e la visualizzazione dei dati. Con SciPy un programma Python può diventare un ambiente per processare dati e sistemi di prototipazione con funzionalità pari a quelle offerte da software come MATLAB, IDL, Octave.

Questa libreria è stata molto utile per la realizzazione delle correlazioni, in quanto fornisce comandi semplici per correlare colonne di DataFrame con diversi algoritmi di correlazione, tra i quali:

pearsonr[14]: metodo che consente di calcolare la correlazione di insiemi di dati mediante l'indice di correlazione di Pearson e i p-value corrispondenti. Il coefficiente di correlazione di Pearson misura la relazione lineare tra due data set. Questo coefficiente di correlazione varia tra -1.0 e 1.0, con il valore 0 a indicare una assenza di correlazione. Il valore 1 indica una relazione lineare completamente positiva, mentre il valore -1 indica una correlazione lineare completamente negativa.

spearmanr[15]: metodo che consente di calcolare la correlazione di insiemi di dati mediante l'indice di correlazione di Spearman e i p-value corrispondenti. La correlazione di Spearman è una misura non parametrica della monotonicità di una relazione tra due data set. Al contrario della correlazione di Pearson, la correlazione di Spearman non assume che i data set siano normalmente distribuiti. Come gli altri coefficienti di correlazione questo varia tra -1.0 e 1.0, con il valore 0 a indicare una assenza di correlazione. Valori pari a -1 o 1 implicano una relazione monotona perfetta. Correlazioni positive implicano che se x cresce, anche y cresce.

Correlazioni negative implicano che se x cresce, y decresce.

Il *p-value* indica approssimativamente la probabilità che sistemi non correlati producano un data set con una correlazione estrema almeno come quella prodotta da questi data set.

2.4 Pandas

2.4.1 Introduzione a Pandas

Pandas[16] è un pacchetto Python che provvede strutture veloci, flessibili ed espressive per rendere le operazioni sui dati facili e intuitive. Mira a essere il building block di alto livello fondamentale per effettuare data analisi in Python.

Pandas consente di analizzare diversi tipi di dati:

- Dati tabellari con colonne eterogenee come sheet Excel e tabelle SQL;
- Serie temporali ordinate e non (non necessariamente fissate in frequenza);
- Dati matriciali (di tipo omogeneo o eterogeneo) con indici di riga e colonna;
- Qualsiasi forma di data set statistici e osservativi. I dati non necessitano di essere etichettati per essere inseriti nelle strutture dati di Pandas.

Pandas è costruito sulla libreria NumPy così da poter integrare correttamente un ambiente scientifico di computazione con tante librerie di terze parti.

Pandas consente di:

- Maneggiare in modo semplice i dati mancanti (rappresentati come NumPy NaN), per dati di qualsiasi tipo;
- Scalabilità: colonne e righe possono essere inserite e rimosse dai DataFrame;
- Allineamento automatico ed esplicito dei dati: gli oggetti possono essere allineati a un set di etichette o usati ignorando le etichette, permette anche di allineare automaticamente I dati provenienti da Series e DataFrame durante i calcoli;
- Consente in modo efficiente e flessibile di effettuare raggruppamenti per effettuare operazioni di split, apply, combine sui set di dati, per aggregare e/o trasformare i dati;
- Consente di convertire dati sporchi e/o indicizzati in modo eterogeneo in altre strutture dati di Python o NumPy;
- Slicing intelligente basato sulle etichette, indicizzazione e creazione di sottoinsiemi di grandi set di dati;
- Merge e join intuitive dei diversi set di dati;
- Possibilità di modellare e ruotare i set di dati;
- Etichettatura gerarchica degli assi;
- Strumenti robusti per l'I/O dei file CSV, Excel, SQL.

2.4.2 Formato dei dati

Pandas utilizza principalmente due strutture dati:

Series: struttura dati del tutto simile a un array a una dimensione con degli indici di riga che possono essere diversi da i soliti numeri usati negli array, infatti è possibile l'utilizzo di stringhe come indice di riga.

DataFrame: struttura dati nel quale si possono avere più colonne per ogni indice, la struttura è del tutto simile a una tabella nella quale si hanno delle colonne e delle righe. Questa è la struttura principale usata da Pandas, su di essa possono essere realizzate operazioni di ogni tipo, come ad esempio estrazioni di righe e/o colonne (le quali vengono estratte come Series), riordinamento, operazioni sugli elementi di riga o di colonna o sulle singole celle, re-indicizzazioni, creazione di pivot-table.

2.4.3 Lettura dei dati

Pandas offre molti comandi per la lettura dei dati che si differenziano per la struttura del file che contiene i dati (csv, xls, sql…) e per il formato della variabile che andrà a contenere il dato. Generalmente i dati letti vengono letti vengono inseriti in un DataFrame di Pandas il quale può essere modellato a seconda delle necessità delle successive operazioni che verranno fatte. Quando si va a leggere dati scritti su file, i quali generalmente sono tabelle, possiamo rinominare righe e colonne, andare a selezionare la colonna che costituirà l'indice di riga, ecc...

Nel caso di dati mancanti in Pandas è possibile individuarli e operare su di essi in quanto sono identificati con il valore NumPy NaN. Pandas nelle sue funzionalità può evitare, sostituire o cancellare righe e/o colonne che contengono parametri nulli.

2.4.4 Gestione dei dati

I DataFrame ottenuti o creati possono essere maneggiati come le tabelle SQL, vi è la possibilità di aggiungere e rimuovere colonne; vi è la possibilità di effettuare delle associazioni tra i diversi DataFrame; effettuare filtraggi e raggruppamenti (Figura 2-1 DataFrame groupby), andando a scegliere colonne o righe specifiche in base all'indice di queste ultime.

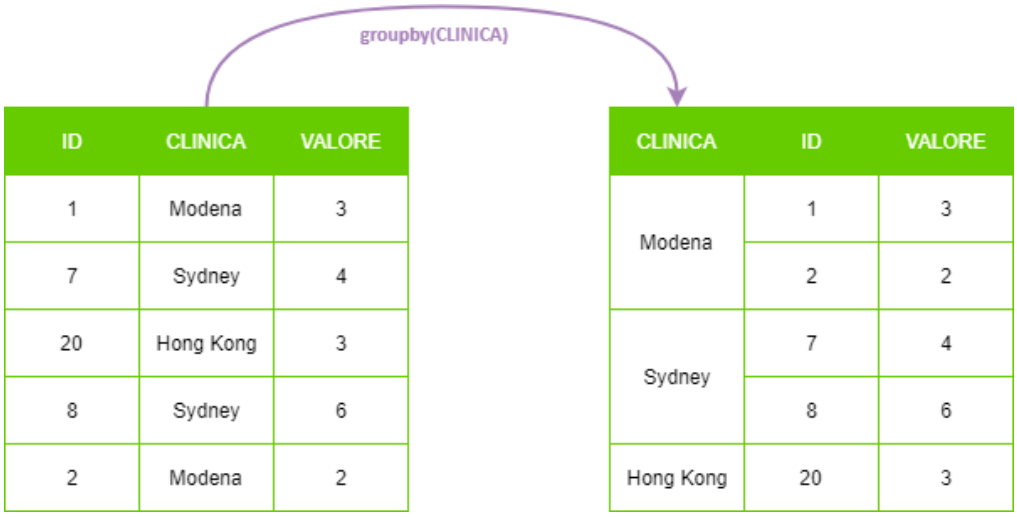


Figura 2-1 DataFrame groupby

Inoltre, proprio come per le tabelle SQL si possono andare a unire DataFrame diversi per ottenere correlazioni di dati con caratteristiche in comune (Figura 2-2 DataFrame merge).

ID	MESE	VALORE A
1	1	3
7	1	4
1	2	3
8	1	6
9	2	2

ID	CLINICA	VALORE B
1	Modena	4
7	Sydney	6
8	Hong Kong	5
9	Sydney	9
2	Modena	4

merge(on='ID')

ID	MESE	VALORE A	CLINICA	VALORE B
1	1	3	Modena	4
7	1	4	Sydney	6
1	2	3	Modena	4
8	1	6	Hong Kong	5
9	2	2	Sydney	9
2			Modena	4

Figura 2-2 DataFrame merge

La possibilità di effettuare raggruppamenti di dati consente di andare a effettuare operazioni su questi gruppi come somme, sottrazioni conteggi e vari calcoli statistici i cui risultati verranno inseriti in colonne specifiche con nome dell'operazione effettuata.

Tra le varie operazioni importanti effettuabili sui dati è presente quella che va a creare le cosiddette pivot-table (Figura 2-3 DataFrame pivot_table) le quali sono tabelle con un indice, l'espansione degli elementi di una colonna come colonne e dei valori legati ad altre colonne come valori della tabella. Questo genere di tabelle risulta essere molto

utile per le rappresentazioni grafiche che verranno poi effettuate sui dati. Le pivot-table sono inserite in DataFrame.

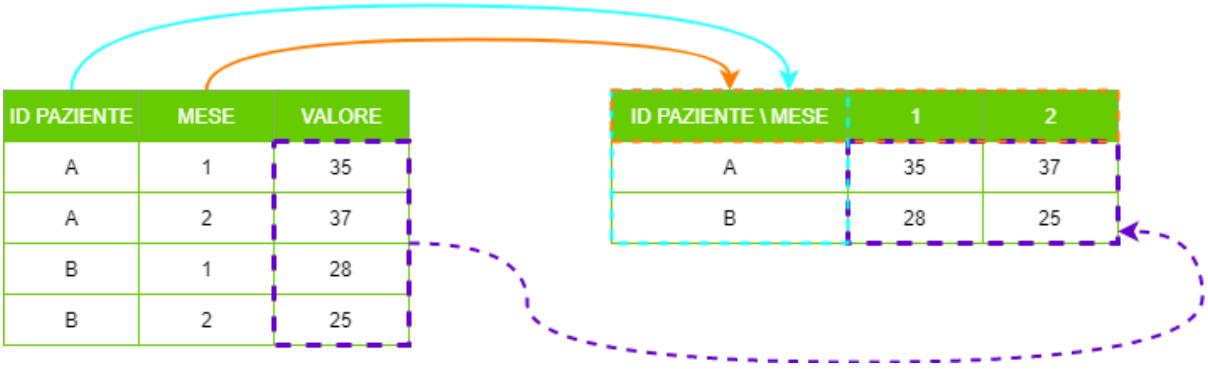


Figura 2-3 DataFrame pivot_table

Rappresentando il data set con una pivot-table, si è potuto evidenziare in modo efficace l'andamento di un valore di un paziente nel tempo, utilizzando una riga sola.

Si puo effettuare l'operazione contraria con il comando stack.

2.5 Salvataggio dei dati

I dati letti da file (csv, xls, sql, …) e/o creati da un processo, possono essere usati per creare file (csv, xls, sql, …) o grafici salvabili su file immagine.

2.6 Rappresentazione dei dati e dei risultati

Siccome molti dei dati sono difficilmente comprensibili così come sono, sia perché sono in grandi quantità, sia perché spesso senza poterli confrontare con altri, è difficile

comprenderne il significato, risulta molto efficace prendere questi dati e utilizzarli per la costruzione di grafici.

Sono presenti diverse librerie che consentono la rappresentazione dei dati in Python, quelle usate per questo progetto sono: Matplotlib, Seaborn e Plotly.

2.7 Matplotlib

Matplotlib[17] è una libreria per la creazione di plot 2D di array Python. Questa libreria è nata per emulare i comandi grafici offerti dal linguaggio MATLAB, essa però è indipendente da MATLAB è può essere usata in modo più Pythonico.

Matplotlib è scritta completamente in Python e fa uso delle funzionalità della libreria NumPy e di altre librerie per offrire buone performance all'aumentare delle dimensioni.

Matplotlib è stata creata con la filosofia che il programmatore deve poter creare plot semplici in pochi comandi, senza dover creare oggetti e invocarne i metodi.

Il codice della libreria Matplotlib è concettualmente diviso in tre parti:

- L'interfaccia pylab che consente all'utente di creare plot con un codice simile a quello usato in MATLAB;
- Il frontend di Matplotlib è composto da renderer che creano e maneggiano le figure, il testo, le linee, i plot e altri oggetti. Questa è un'interfaccia astratta che non conosce l'output;
- I dispositivi backend anche detti renderer, trasformano il frontend rendendolo visibile sul dispositivo.

2.8 Seaborn

Seaborn[18] è una libreria per creare grafici statistici in Python. Essa è costruita sulla libreria Matplotlib e integrata con le strutture Pandas.

Le funzionalità offerte da Seaborn sono:

- API orientate ai data set per esaminare le relazioni tra multiple variabili;
- Supporto all'uso di variabili categoriche per visualizzare osservazioni o aggregazioni statistiche;
- Stime automatiche e plotting di modelli di regressione lineare per diversi tipi di variabili;
- Viste utili sulla struttura complessiva di un set di dati complessi;
- Alto livello di astrazione per la creazione di griglie di plot multipli per la costruzione di visualizzazioni complesse;
- Controllo conciso dello stile delle figure create da Matplotlib con l'ausilio di temi grafici;
- Strumenti per la scelta e creazione delle palette di colori.

Seaborn vuole far sì che la visualizzazione sia una parte centrale dell'esplorazione e della comprensione dei dati. Le sue funzionalità orientate ai data set forniscono funzionalità per operare su DataFrame e array contenenti interi data set, effettuando internamente le necessarie operazioni semantiche e le aggregazioni statistiche necessarie per produrre plot informativi.

2.9 Rappresentazione statica

Per la rappresentazione statica dei dati di questo studio è stata scelta la libreria Matplotlib.pyplot la quale è completata dalla libreria Seaborn i cui risultati sono di migliore appeal.

Attraverso queste librerie si possono rappresentare grafici di diverso tipo, quelli da me utilizzati sono stati:

Istogrammi: grafico costituito da rettangoli adiacenti le cui basi sono allineate su un asse dotato di nomi per ognuno dei rettangoli (Figura 2-4 Seaborn barplot), il quale definisce cosa rappresenta ogni singolo rettangolo. L'altro asse definisce invece una misura di quantità della caratteristica descritta da ognuno dei rettangoli. Questo tipo di grafico è utile per mettere in relazione le quantità;

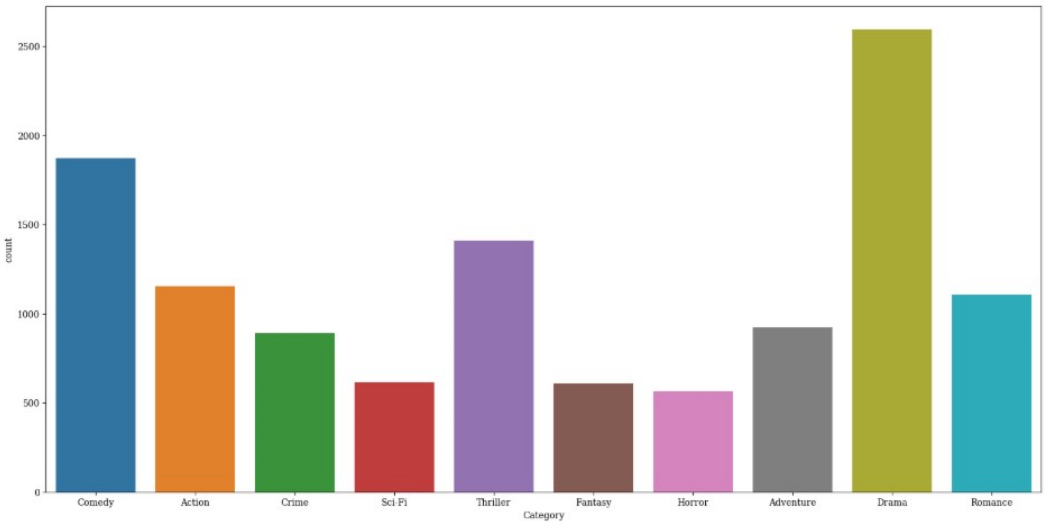


Figura 2-4 Seaborn barplot

- *Heatmap:* rappresentazione grafica dei dati dove i singoli valori contenuti in una matrice sono rappresentati da colori (Figura 2-5 Seaborn heatmap). Con questo tipo di grafico le variazioni dei dati risultano molto evidenti, in quanto sono rappresentate in tabelle nelle quali vengono associate scale di colori alle variazioni dei valori;

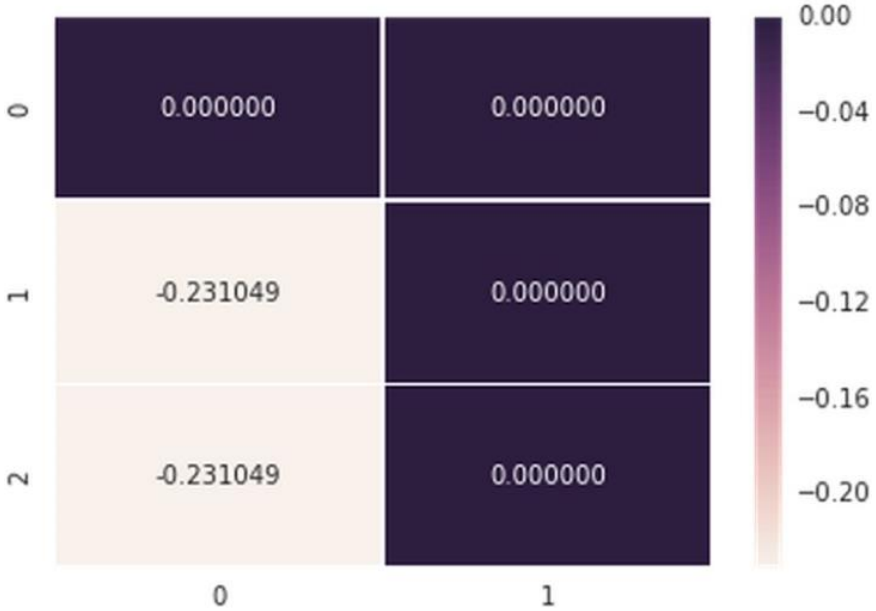


Figura 2-5 Seaborn heatmap

Siccome i grafici sono statici, l'unico modo di leggerli è quello di scrivere i valori delle caselle in esse, in quanto il colore assunto dalla tabella non fornisce un dato preciso.

2.10 Plotly

Plotly[19] è una libreria grafica per Python e altri linguaggi che consente la creazione di grafici di alta qualità e interattivi online e offline. Con questa libreria si possono creare grafici di qualunque tipo: line plot, scatter plot, area chart, bar chart, error bar, box plot, histogram, heatmap, subplot, multiple-axes, polar chart, bubble chart.

Figura 2-6 Esempi di plots con Plotly

Si può quindi definire Plotly come una toolbox per la visualizzazione. Al di sotto di ogni grafico vi è un oggetto JSON, il quale ha una struttura a dizionario. Cambiando i valori di alcune parole chiave in questo oggetto, è possibile creare plot differenti e più dettagliati.

2.11 Dash

Dash[20] è una libreria degli stessi sviluppatori della libreria Plotly, con la quale si integra.

Nel dettaglio Dash è un framework Python per la creazione di applicazioni web.

Dash è scritto sulle librerie Flask, Plotly.js e React.js, risulta ideale per costruire applicazioni per la visualizzazione di dati consentendo una forte interazione con l'utente.

Dash, con i suoi semplici pattern, consente di astrarre da tutti gli aspetti e i protocolli richiesti per la costruzione di applicazioni web-based.

Le applicazioni create con Dash sono renderizzate dai browser web e possono essere distribuite mediante URL. Siccome sono visualizzate sul browser web, sono completamente indipendenti dall'architettura del sistema.

Grazie a questa libreria è quindi possibile creare web app con le quali interagire con i plot.

L'interazione con i plot avviene in modo semplice e intuitivo per l'utente, infatti si possono predisporre dei menu a tendina, dei selettori a slitta, dei selettori di range, dei radio button e tanti altri con i quali si vanno ad attivare delle routine che possono essere utilizzate per modificare i dati plottati e/o il modo in cui vengono plottati.

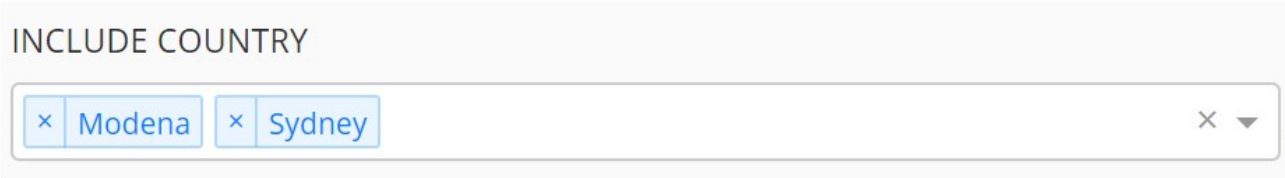


Figura 2-7 Dash scelta multipla

Figura 2-8 Dash selettore di range

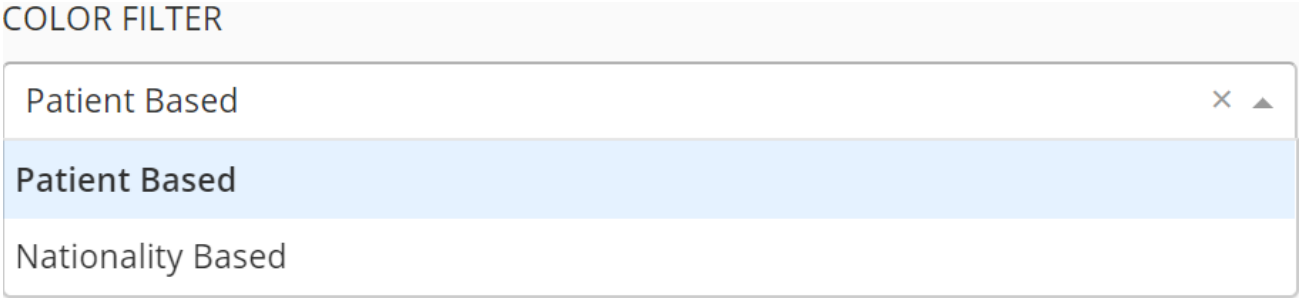


Figura 2-9 Dash dropdown menu

Ogni volta che viene modificato uno o più parametri in questi menu a disposizione dell'utente, viene effettuata una chiamata a tutte le funzioni che nell'apposito decoratore hanno l'oggetto con l'id dell'oggetto modificato.

2.12 Rappresentazione dinamica

Per la rappresentazione dinamica e interattiva dei dati ho utilizzato la libreria Plotly e la libreria Dash la quale consente di rappresentare grafici di diverso tipo in pagine html.

Proprio perché si utilizzano pagine html e non file immagine, risulta possibile l'interrogazione di tali grafici, rendendo semplice e intuitiva la loro lettura.

Mediante la libreria Plotly ho realizzato molti dei grafici relativi al secondo batch di dati e indici.

Ho realizzato grafici per gli indici sotto forma di heatmap, più immediata nel visualizzare variazioni importanti e confrontare l'andamento dei diversi indici per ognuno dei pazienti, sotto forma di grafico a linee, utile per visualizzare le differenze tra i vari pazienti e l'andamento nel tempo.

Ho poi realizzato grafici a linee per i parametri adibiti alla creazione degli indici consentendo così il confronto di questi dati tra i diversi pazienti nei diversi periodi, rendendo possibile l'individuazione dei parametri che effettivamente sono utili e quelli che per le minime o nulle variazioni nei periodi di tempo non lo sono.

2.13 Cosa si intende per interattività

I grafici creati da questa libreria possono essere interattivi, vi è infatti la possibilità di esplorare i grafici, selezionare, ingrandire, rimpicciolire e interrogare gli elementi semplicemente puntandoli con il cursore del mouse.

Inoltre, in alcuni tipi di grafici vi è la possibilità di escludere elementi o selezionarne solo alcuni.

Nel caso della heatmap seguente (Figura 2-10 Plotly heatmap) risulta complesso individuare il mese in quanto i colori sono molto simili, ma siccome il grafico è interattivo il mese e altri valori vengono visualizzati esplicitamente.

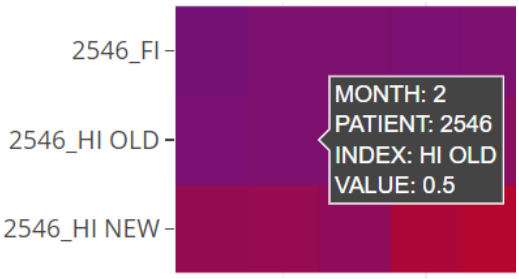


Figura 2-10 Plotly heatmap

Il problema dei plot non interattivi diventa ancora più evidente nei grafici a linee, come quello seguente (Figura 2-11 Plotly lineplot), nei quali non è presente lo spazio per poter inserire le specifiche di un determinato punto nel grafico e nei quali per il grande numero di dati spesso i colori si ripetono o si sovrappongono.

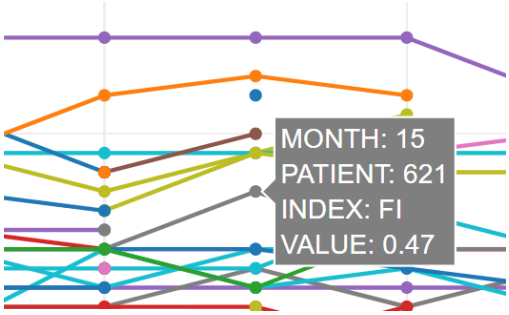


Figura 2-11 Plotly lineplot

Capitolo 3: Progetto MySAwH

In questo capitolo vado a descrivere lo studio MySAwH, partendo dalle sue caratteristiche principali, analizzando via via gli obbiettivi che si è posto nell'analisi dello stato di salute di pazienti anziani affetti da HIV.

3.1 Introduzione al progetto MySAwH

L'Organizzazione Mondiale della Sanità (OMS) è alla continua ricerca di strumenti per lo studio dell'invecchiamento sano. My Smart Age with HIV (MySAwH)[21] è uno studio prospettico in corso, coordinato dal Prof. Giovanni Guaraldi, progettato per consentire agli anziani (di età> 50 anni) di ottenere un invecchiamento sano, attraverso lo studio di pazienti che convivono con l'HIV in Italia, Australia e Hong Kong. Lo studio coinvolge circa 280 pazienti monitorati per 24 mesi, sono stati scelti pazienti malati di HIV in quanto questa patologia comporta un invecchiamento precoce. I dati raccolti sono altamente eterogenei per quanto riguarda il tipo, l'origine e il tasso di acquisizione e includono: la valutazione geriatrica completa raccolta dagli operatori sanitari durante le visite di studio; conteggio passi, ore di sonno e calorie raccolte quotidianamente da un dispositivo indossabile; risultati di questionari su capacità funzionali e qualità della vita, registrati tramite una app dedicata per smartphone (MySAwHApp).

La finalità di questo studio è di seguire un approccio basato sui dati per standardizzare un indice di capacità intrinseca (ICI) in relazione all'età e ai risultati sanitari rilevanti, tra cui la fragilità e lo stato di HIV[22].

3.1.1 Obbiettivi di questo studio

Questo studio è volto alla responsabilizzazione dei pazienti anziani affetti da HIV attraverso la promozione di una vita salutare, monitorata e valutata con l'ausilio della applicazione MySAwH.

Questa applicazione mobile è progettata per consentire agli utenti che ne fanno uso di poter valutare il proprio stato di invecchiamento.

Le valutazioni ottenute dagli utilizzatori vengono utilizzate per definire un indice di fragilità (FI) e un indice di capacità intrinseca del paziente (IC).

3.1.2 Dimensione dello studio

Nello studio in questione sono stati coinvolti circa 280 pazienti con età superiore ai 50 anni, dei quali, circa 150 sono Italiani, 90 sono Australiani e 30 provengono da Hong Kong. I pazienti sono stati monitorati per un periodo di 24 mesi, attuando diverse procedure tra le quali:

Obbiettivi primari:

- Terapie Antiretrovirali sulla infezione da HIV (ART);
- Monitorare l'attività fisica con i tracking device;
- Spiegare ai pazienti come utilizzare l'app di MySAwH.

Visite mediche:

- Effettuate all'inizio dello studio, dopo un anno (+9 mesi) e dopo due anni (+18 mesi);
- Valutazione geriatrica attraverso l'indice di fragilità.

3.1.3 Applicazione MySAwH

L'applicazione interagisce con l'utente attraverso una interfaccia amichevole di un "Health Coach" che invierà al paziente messaggi riguardanti la transizione della fragilità avvenuta nel mese precedente e la promozione di uno stile di vita sano.

3.2 Cosa sono gli indici di fragilità e di capacità intrinseca?

3.2.1 Da cosa derivano gli indici?

I due indici sono generati mediante variabili di salute che comprendono dati di funzione fisica (PF-D), raccolti da dispositivi di tracciamento fitness, dati di valutazione momentanea ecologica (EMA-D), raccolti attraverso dei questionari e dati sanitari del medico (PH-D), raccolti durante le visite dei pazienti.

Le variazioni che subiscono gli indici nel tempo descrivono lo stato del paziente, infatti sebbene la salute generalmente peggiori con l'età, la relazione tra l'invecchiamento, la salute fisica e quella psicologica è un valore dinamico nel quale l'impegno dei pazienti ad avere un migliore stile di vita può portare a notevoli cambiamenti nei valori degli indici.

3.2.2 Indice di fragilità (FI)

Siccome la salute peggiora con l'età, per comprendere la complessità dell'invecchiamento e della fragilità dell'infezione cronica da HIV si è arrivati a costruire un indice di fragilità basato su 43 variabili, ognuna delle quali rappresenta un deficit di salute del paziente. Ogni deficit viene selezionato per essere associato a esiti avversi per

la salute e con l'età, avere una prevalenza di almeno l'1% e non diventare universale con l'età.

Le variabili utilizzate sono di diverso tipo booleano, intero, decimale; per poterle utilizzare nel calcolo degli indici sono state opportunamente convertite a valori booleani che hanno un valore pari a 1 se il deficit è completamente espresso e 0 se il deficit è assente. La percentuale dei deficit accumulati da un individuo consente di valutare lo stato della fragilità dell'individuo stesso facendo il rapporto tra i deficit individuati e il numero di variabili usate per il calcolo di questi deficit[23].

Con i risultati ottenuti si è potuto confermare che valori bassi di FI dimostrano che l'individuo sta avendo un invecchiamento cognitivo di successo mentre valori alti di FI identificano individui che sono a rischio di morte o con Multi-Morbosità. Per Multi-Morbosità si intende la presenza simultanea su un paziente di 2 o più delle seguenti co-morbosità: ipertensione, diabete, problemi cardiovascolari, disfunzioni renali, osteoporosi, cancro, cirrosi epatica, ipotiroidismo, COPD, ipogonadismo.

3.2.3 Indice di fragilità senza multi-morbosità (FI NO MM)

In un secondo momento questo indice è stato rivisto, con la conseguente rimozione delle variabili legate alle multi-morbosità, questa modifica lo ha portato a 34 variabili.

3.2.4 Indice di capacità intrinseca (IC)

Oltre all'indice di fragilità è presente un secondo indice detto Intrinsic Capacity[24] o di capacità intrinseca il quale è generato mensilmente, integrando i dati ottenuti da parametri fisiologici (passi, energie spese e durata del sonno), raccolti attraverso dispositivi indossabili ed electronic Patient Reported Outcomes (ePRO) raccolti utilizzando gli smartphone.

Gli ePRO sono raccolti ogni 48 ore sottoponendo al paziente una o più domande provenienti da questionari che coprono differenti ambiti, tra i quali: vulnerabilità sociale,

alimentazione, livello di stress, attività fisica, umore, abitudini e comportamento neuro cognitivo.

Queste domande vengono fatte almeno uno volta al mese così da avere un indice sempre aggiornato per gestire i dati utilizzati per la creazione dei due indici è stata creata una applicazione web e una app per dispositivi mobili.

L'IC è quindi la composizione di tutte le capacità mentali e fisiche di un individuo, misurabili in ogni momento della sua vita. A livello di popolazione vi è una tendenza generale a declinare l'IC dalla seconda metà dell'età adulta in poi, in quanto per la maggior parte degli individui questa tendenza non sarà liscia, ma rifletterà più battute di arresto e recuperi potenziali.

Nelle popolazioni nelle quali la morte di solito si verifica in età avanzata, verso la fine della vita la maggior parte delle persone subisce perdite significative nei valori dell'IC. La conservazione delle capacità mentali e/o fisiche in alcuni ottantenni può significare che loro hanno un IC complessivo più spesso associato a persone molto più giovani.

Quindi l'età cronologica è inadeguata per definire una persona come a rischio di eventi negativi.

Gruppo	Variabili Indici	Provenienza	non-MMFI (34)	MMFI (27)	IC (27)
body composition	Lipoatrophy Multicenter AIDS Cohort Study (MACS) criteria > 1	visita	yes		
body composition	High or low BMI <18 or >25 kg/m2	visita	yes	yes	
body composition	High waist circumference If female: >88 cm, If male: >102 cm	visita	yes		
fasting blood test including metabolic and HIV variables	High total cholesterol >200 mg/dl	visita	yes	yes	
fasting blood test including metabolic and HIV variables	High low-density lipoprotein >100 mg/dl	visita	yes		
fasting blood test including metabolic and HIV variables	Low high-density lipoprotein <40 mg/dl	visita	yes		
fasting blood test including metabolic and HIV variables	High triglycerides >150 mg/dl	visita	yes		
fasting blood test including metabolic and HIV variables	Abnormal white blood cell counts <4000 cells/microlitri	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Anemia (female: <10gr/dl; male < 12 gr/dl)	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Hepatitis C coinfection Positive	visita	yes		
fasting blood test including metabolic and HIV variables	Hepatitis B coinfection Hepatitis B antigen positive	visita	yes		
fasting blood test including metabolic and HIV variables	Vitamin D insufficiency <30 ng/ml	visita	yes		
fasting blood test including metabolic and HIV variables	Abnormal parathyroid hormone >60 pg/ml	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated D-dimer >Sample mean (358)	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated C-reactive protein >0.7 mg/l	visita	yes		
fasting blood test including metabolic and HIV variables	Hyponatremia <125 mmol/l	visita	yes		
fasting blood test including metabolic and HIV variables	Proteinuria or albuminuria >5 mg/dL	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated aspartate transaminase >31 U/l	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated alanine transaminase >31 U/l	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Abnormal alkaline phosphatase <38 or >126 U/l	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated g-glutamyl transphosphatase >55 U/l	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Low platelets <150 billion/l	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Abnormal potassium <3.5 or >5.3 mEq/l	visita	yes		
fasting blood test including metabolic and HIV variables	Abnormal phosphorus <2.5 or >5.1 mg/dl	visita	yes		
fasting blood test including metabolic and HIV variables	Abnormal thyroid-stimulating hormone <0.27 or >4.2 mIU/l	visita	yes		
fasting blood test including metabolic and HIV variables	Elevated total bilirubin >1.10 mg/dl	visita	yes	yes	
fasting blood test including metabolic and HIV variables	creatinina	visita		yes	
fasting blood test including metabolic and HIV variables	Glucosio	visita		yes	
fasting blood test including metabolic and HIV variables	Unintentional weight loss of more than 4.5 Kg in the previous year	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Pressione sistolica	visita	yes	yes	
fasting blood test including metabolic and HIV variables	Pressione diastolica	visita	yes	yes	
Co-morbidities	Cardiovascular disease (clinical diagnosis)	visita		yes	
Co-morbidities	Hypertension (blood pressure or treatment)	visita		yes	
Co-morbidities	Diabetes type II (fasting glucose > 125md/dl or treatment)	visita		yes	
Co-morbidities	CKD (two estimated egfr < 60ml /min/)	visita		yes	
Co-morbidities	Cirrohosis (FIB 4 score > 3.25)	visita		yes	
Co-morbidities	COPD (FEV1/FVC < 0.7)	visita		yes	
Co-morbidities	Osteoporosis (DXA z score < -2.5)	visita		yes	
Co-morbidities	Any Cancer (clinical diagnosis with biopsy confirmation)	visita		yes	
Polypharmacy	Polypharmacy (> 5 drug classes excluding antiret therapy)	visita	yes	yes	

Figura 3-1 Requisiti clinici di progetto

Gruppo	Variabili Indici	Provenienza	non-MMFI (34)	MMFI (27)	IC (27)
Patient related outcomes	Unemployment Self-report	visita App			yes
EMA	EMA physical activity	survey			yes
EMA	EMA mood (NEGATIVE) domanda 14	survey			yes
EMA	EMA mood (NEGATIVE) domanda 15	survey			yes
EMA	EMA mood (NEGATIVE) domande 11,12,13,16,17,18,19,20	survey			
EMA	EMA mood (POSITIVE) domanda 4	survey			yes
EMA	EMA mood (POSITIVE) domanda 10	survey			yes
EMA	EMA mood (POSITIVE) domande 1,2,3,5,6,7,8,9	survey			
EMA	EMA stress level	survey			yes
EMA	EMA social (home)	survey			yes
EMA	EMA social (alone)	survey			yes
EMA	EMA social (socialization)	survey			yes
EMA	EMA eating (binge) domanda 1	survey			yes
EMA	EMA eating (binge) domanda 2	survey			
EMA	EMA eating (craving for food) domanda 8	survey			yes
EMA	EMA eating (craving for food) domande 3-4-5-6-7	survey			
EMA	EMA eating (loss of appetite) domanda 9	survey			yes
EMA	EMA sleep quality	survey			yes
EMA	EMA sleep duration	survey			yes
EMA	EMA smoke	survey			yes
IPAQ	IPA sitting	survey			yes
IPAQ	IPA walking activity 1	survey		yes	yes
IPAQ	IPA walking activity 2	survey			
IPAQ	IPA moderate activity 1	survey			yes
IPAQ	IPA moderate activity 2	survey			
IPAQ	IPA intensive activity 1	survey			yes
IPAQ	IPA intensive activity 2	survey			
SOCIAL VULNERABILITY	SHARE 1	survey			
SOCIAL VULNERABILITY	SHARE 2	survey			
SOCIAL VULNERABILITY	SHARE 2A	survey			yes
SOCIAL VULNERABILITY	SHARE 3	survey			
SOCIAL VULNERABILITY	SHARE 4	survey			
SOCIAL VULNERABILITY	SHARE 5	survey			
SOCIAL VULNERABILITY	SHARE 5A	survey			yes
NEUROCOGNITE	NEURO 1	survey			yes
NEUROCOGNITE	NEURO 2	survey			
NEUROCOGNITE	NEURO 3	survey			yes
GARMIN	calories	braccialetto			yes
GARMIN	steps	braccialetto			yes
GARMIN	ore dormite	braccialetto			yes
EMA (Mood)	Bothered (Irritable)	survey	yes	yes	
SHARE	Happy (How satisfied are you with your life in general)	suervey	yes	yes	
SHARE	Lonely (I felt lonely)	suervey	yes	yes	
OpenClinica - EMA (EAT)	Poor appetite (Do you have a decrease in appetite)	survey	yes	yes	

Figura 3-2 Requisiti psicologici di progetto

3.3 Analisi statistiche

L'obbiettivo primario è quello di andare a correlare l'indice di Capacità Intrinseca con l'Indice di Fragilità, cercando di comprendere le motivazioni che scatenano cambiamenti nei due indici e cosa accade all'altro indice quando uno dei due cambia.

Inoltre, dovranno essere correlati i valori degli indici con alcuni score relativi allo stato di salute del paziente:

- *IADL*: [25] scala che tiene conto di funzioni fisiche più complesse, è costituita da un elenco di 8 attività funzionali complesse e per ciascuna di esse sono descritti 3 o più livelli di competenza. Le attività comprendono:
    1) usare il telefono;
    2) fare la spesa;
    3) cucinare;
    4) fare le pulizie di casa;
    5) lavare la biancheria;
    6) autonomia di trasporto;
    7) gestione indipendente della terapia farmacologica;
    8) autonomia nella gestione delle questioni economiche.

  Ad ogni risposta viene attribuito un punteggio. Quanto più il punteggio è alto tanto più il paziente è compromesso dal punto di vista delle attività funzionali contemplate dalla scala IADL. I valori dei punteggi vengono utilizzati per creare il punteggio finale del paziente nel range 0-100.

- *KATZ*:[26] scala di valutazione delle attività di base della vita quotidiana proposta da Katz nel 1963 valuta in modo accurato 6 attività di base:
    1) fare il bagno;
    2) vestirsi;
    3) toilette;

4) spostarsi;
5) continenza urinaria e fecale;
6) alimentarsi.

I punteggi assegnati sono dicotomici (dipendente/indipendente), per questo motivo l'indice KATZ è meno flessibile soprattutto in popolazioni di soggetti fragili, quali ad esempio i dementi. Nonostante la minore flessibilità limiti l'ampia diffusione dell'indice di KATZ, lo rende utile soprattutto per la valutazione del livello di autonomia di larghe popolazioni di individui o per valutazioni in studi longitudinali. L'indice KATZ è poco adatto per la valutazione della risposta ad interventi riabilitativi o assistenziali nel breve/medio periodo.

- *EQ5D5L*: [27] EQ5D è uno strumento standardizzato per misurare lo stato generico di salute. È molto usato in studi clinici, indagini sanitarie, studi economici.

  EQ5D è progettato per l'auto-completamento e, in quanto tale, acquisisce informazioni direttamente dal rispondente, generando in tal modo dati conformi ai requisiti generali di tutte le misure del referto del paziente (PRO). Questo indice generalmente usa 3 variabili, le quali in certe situazioni mostrano delle limitazioni. EQ5D5L è un indice che consente di avere una maggiore precisione nel caso ci siano piccoli cambi di salute e problematiche come schizofrenia, alcolismo, problemi uditivi. Questo indice definisce 3,125(5^5) possibili stati di salute e la sua validità e affidabilità si basa sull'analisi delle morbosità come tumori, diabete di tipo 2, COPD, asma, problemi cardiovascolari, ecc.… ed è molto consigliato per lo studio di pazienti anziani.

- Health: [28] misura dello stato di salute fatta dal paziente basandosi su una scala Likert a 7 punti, dove i valori più alti indicano un migliore stato di salute.

- *Cadute e perdita di equilibrio*: sono valori presi dai questionari fatti ai pazienti al momento delle visite cliniche, le domande utilizzate per questi score sono:

  "In the past 6 months have you had a fall?",

  "In the past 6 months have you been concerned with losing your balance and falling while doing your usual daily activities?"

Parte II: Pulizia, analisi, elaborazione dei dati e visualizzazione di dati e risultati.

Capitolo 4: Operazioni di pulizia dei dati

In questo capitolo vado a descrivere le operazioni di data collection da me effettuate per ripulire i dati in modo tale da renderli utilizzabili nelle operazioni successive. Queste operazioni iniziano con l'analisi dei requisiti di progetto dai quali si possono individuare i dati utili, escludendo quelli non contemplati nel calcolo degli indici.

I problemi principali individuati e risolti sono stati:

- Formato irregolare delle date e dei nomi;
- Presenza del valore '(null)' e del valore 999 nei campi non riempiti;
- Convertire le unità di misura dove serve;
- Presenza di valori anomali;
- Molti parametri non necessari;
- Sostituire gli opportuni valori con dei booleani.

4.1 Analisi dei requisiti

Come prima cosa ho analizzato il file Excel che fornisce i requisiti del progetto, in tale file è presente la tabella mostrata in precedenza (Figura 3-1 Requisiti clinici di progetto, Figura 3-2 Requisiti psicologici di progetto) e nelle altre sheet le spiegazioni del significato di ognuno dei parametri e le modalità per convertirli in valori booleani utili al calcolo degli indici.

4.1.1 Formato irregolare delle date, degli id e dei nomi delle colonne

Le date si presentavano in formati diversi il che rendeva impossibile fare confronti tra di loro per questo motivo le ho uniformate al formato Little-endian di gg/mm/yyyy.

I nomi assegnati ai pazienti nei file clinici sono molti, perciò ho scelto per uniformità con i dati psicologici, di utilizzare i valori contenuti nella colonna subject_id come nomi per i pazienti rinominando tale colonna a id_patient.

Inoltre, alcuni pazienti presentavano uno spazio prima del id, per evitare confusione anch'esso è stato rimosso.

I nomi delle colonne nei file Excel presentavano degli spazi prima del nome, anche in questo caso sono stati rimossi per evitare di fare confusione nelle operazioni successive.

Alcune sheet del file Excel riguardante i dati clinici presentano due volte la riga contenente i nomi delle colonne, il problema è stato risolto utilizzando la seconda riga che nei casi coinvolti risultava essere quella completa di tutti i parametri.

4.1.2 Presenza del valore '(null)' o del valore 999 nei campi non riempiti

La presenza del valore '(null)' o del valore 999 è stata risolta andando a sostituirli nelle loro occorrenze con NumPy NaN.

4.1.3 Convertire le unità di misura dove serve

Confrontandoci con i medici, ci siamo accorti che alcuni parametri clinici presentavano valori strani, e dopo averli divisi per provenienza ci siamo accorti che questi parametri avevano unità di misura differenti da quelle usate nel centro di Modena. Per questo motivo viene applicata una conversione alle variabili coinvolte.

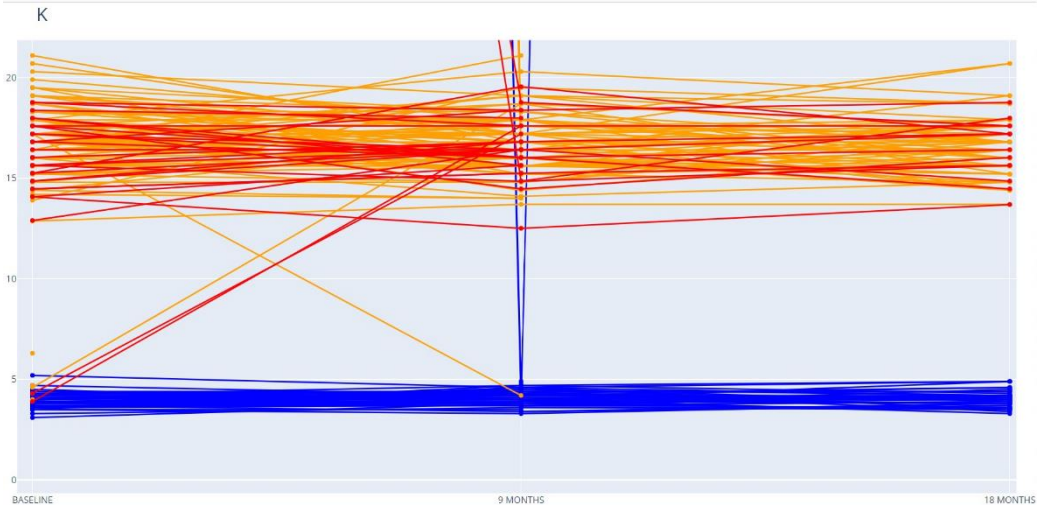


Figura 4-1 Problema delle unità di misura

Nell'immagine (Figura 4-1 Problema delle unità di misura) sono visualizzati i problemi riguardanti le unità di misura per i valori del Potassio (Blu – Modena, Giallo – Sydney, Rosso – Hong Kong).

4.1.4 Presenza di valori anomali

Sempre confrontandoci con i medici abbiamo individuato delle soglie di validità che consentono di rimuovere i valori non ammissibili.

Se il valore di un parametro è all'interno della soglia è accettato altrimenti viene sostituito con il valore NumPy NaN.

4.1.5 Sostituire alcuni valori con valori booleani

Per rendere il tutto più comprensibile alle letture successive ho optato per sostituire valori come 'Positive' e 'Yes' con il valore booleano True e i valori 'Negative' e 'No' con il valore booleano False.

4.1.6 Molti parametri non necessari

La presenza di un gran numero di parametri non necessari per il calcolo porta a dei rallentamenti negli istanti in cui si caricano i dati.

Per risolvere il problema sono andato ad estrarre le colonne necessarie per il calcolo degli indici, scartando le altre.

Queste colonne estratte sono state utilizzate per creare nuovi file Excel:

- Uno per i dati medici e per le prescrizioni di farmaci;
- Uno per i dati psicologici, con diverse sheet per ognuno dei parametri considerati;
- Uno per i dati relativi alla situazione lavorativa dei pazienti nei diversi periodi;
- Uno contenente i dati riguardanti le analisi fatte dal dispositivo Garmin.

4.2 Script per la pulizia dei dati

I seguenti script sono tutti contenuti nel file “Data_Cleaner.py”.

4.2.1 Script per la pulizia dei dati base dei pazienti

Questo script (Script 1 Pulizia dati basici dei pazienti) crea un file con i dati basici dei diversi pazienti: id, anno di nascita, centro di ricerca, da quando ha l’HIV, ecc...

Per creare tale file, apre la sheet SUBJECTS nel file contenente i dati clinici, questa sheet contiene tutti i dati personali del paziente (esclusi nome e cognome per questioni di privacy) e opera le seguenti operazioni:

- Rinomina la colonna ‘subject_id’ a ‘id_patient’ (scelta progettuale per uniformare il nome delle colonne contenenti gli id);
- Rimuove gli spazi presenti in alcuni “id_patient”;

- Uniforma le sigle relative al sesso del paziente a ‘M’ per i pazienti di sesso; maschile e ‘F’ per i pazienti di sesso femminile;
- Ripulisce i valori relativi alle date di nascita.

Infine, salva i dati puliti nel file “PatientBasicData2019.xlsx” all’interno della cartella data.

Segue il codice che effettua le operazioni descritte.

```
def AllPatientsBasicData(file_clinical_data, return_df=False):
    """Methods that clean data keeping only necessary parameters."""
    fcl = file_clinical_data
    fAPBD = 'data/PatientBasicData2019.xlsx'

    subject = pd.read_excel(fcl,
                            sheet_name='SUBJECTS',
                            header=1)[['subject_id',
                                       'study_subject_id',
                                       'subject_date_of_birth',
                                       'site_name',
                                       'subject_sex']]
    # Rename id column
    subject = subject.rename(columns={'subject_id': 'id_patient'})


    # Remove space before id_patient
    subject['id_patient'] = subject['id_patient'].map(
        lambda x: x.strip())

    # Clean subject sex data
    subject['subject_sex'] = subject['subject_sex'].map(
        lambda s: 'M' if s in ['m', 'M'] else 'F')

    # Clean date
    subject['subject_date_of_birth'] =
        clean_dates(subject['subject_date_of_birth'])

    # Write data on file
    writer = pd.ExcelWriter(fAPBD)
    subject.to_excel(writer)
    writer.save()

    # Return df
    if return_df:
        return subject
```

Script 1 Pulizia dati basici dei pazienti

4.2.2 Script per la pulizia dei dati clinici

Questo script (Script 8 Pulizia dati clinici) è necessario per la pulizia dei dati clinici, questo gruppo di dati è quello che presenta la maggior parte dei problemi.

Lo script apre la sheet relativa alle visite, quella relativa alla registrazione dei pazienti, quella relativa ad alcuni dati medici aggiunti in un secondo momento (pressione e perdita di peso), e quella relativa all'assunzione di farmaci e ne preleva le colonne necessarie per il calcolo degli indici. Di seguito verifica la presenza del file relativo ai dati basici dei pazienti, se esiste lo apre, altrimenti invoca lo script che lo crea.

Per ripulire i dati opera come segue:

- Ripulisce le date relative alle visite, all'inizio del HIV e della terapia antiretrovirale;
- Unisce i tre datasheet caricati utilizzando il comando merge di Pandas;
- Rimuove gli spazi presenti nei titoli delle colonne;
- Rimuove le righe '(null)' nel DF relativo ai farmaci assunti;
- Aggiunge al DF relativo ai farmaci assunti l'"id_patient" associato allo "study_subject_id" (un secondo identificativo usato per i dati clinici);
- Cancella la colonna relativa allo "study_subject_id" nel DF relativo ai dati clinici e in quello relativo ai farmaci assunti;
- Per ogni singolo parametro (colonna del DF), rimpiazza i valori '(null)' e 999 con il valore -1, se il parametro necessita operazioni di pulizia (se è nella lista "ClinicalParameterToClean"):
    - Converte la colonna in valori decimali (astype(float));
    - Invoca la funzione corrispondente nel file "parameterClinicClean.py", mediante il comando eval, tale funzione effettuerà le operazioni necessarie a ripulire tale colonna, se una delle righe della colonna presenta valori invalidi rimpiazza il valore con -2;

- Ripulisce le colonne con valori simil-booleani (il nome della colonna finisce con la stringa 'label'), convertendo i valori 'Positive' e 'Yes' a True e 'Negative' e 'No' a False.

I valori -1 e -2, non concepibili come valori medici, sono usati per creare il file con i conteggi dei dati relativi ai pazienti. Questo file avra per ogni evento-centro-parametro il conto dei valori buoni, cattivi e assenti. Il file che contiene tali parametri è il file "Clinic_Counts2019.xlsx". Sostituiti i valori -1 e -2 con il valore NumPy NaN, salva il DF relativo ai dati clinici e quello relativo ai farmaci assunti dai pazienti nel file "Clinic_clean_data2019.xlsx" in due sheet separate.

Segue uno spaccato del codice nel quale vengono effettuate le principali operazioni di pulizia.

```
                                    .
                                    .
                                    .
# Remove space before column titles
datac.columns = [c.lstrip() for c in datac.columns]

# Drop drugs null
datad = datad[datad.Drugs != '(null)']
# Merging drugs data to subject to get id_patient and site_name
datad = pd.merge(subject[['site_name',
                          'id_patient',
                          'study_subject_id']],
                 datad,
                 on='study_subject_id')

# Drop study_subject_id from tables
datad = datad.drop(columns=['study_subject_id'])
datac = datac.drop(columns=['study_subject_id'])

# Remove value 999 and '(null)'
datac = datac.applymap(
            lambda x: -1 if str(x) in ['999', '(null)'] else x)


# Method is applied on numeric columns
for name in list(datac.columns)
    if name in ClinicalParameterToClean:
        # Clean data
        datac[name] = eval('clean_' + name
            + '(datac[name].astype(float))')

# Remove outliers in Yes/No columns
for name in list(datac.columns):
    if 'label' in name:
        datac[name] = datac[name].map(
```

```
                lambda x: True if 'Yes' in str(x)
                                or 'Positive' in str(x)
                                else False if 'No' in str(x)
                                or 'Negative' in str(x) else -1)
                                .
                                .
                                .
```

4.3 Script per la pulizia dei dati psicologici

Questo script (Script 9 Pulizia dati psicologici) è nettamente più articolato, in quanto, data la diversa natura dei dati, si è reso necessario analizzare le varie sheet caso per caso.

Questo script apre il file contenente i dati psicologici, di seguito per ognuna delle sheet di questo file Excel:

- Apre la sheet in un DataFrame;
- Rimuove le colonne non necessarie;
- Utilizzando le funzioni nel file "ParameterPsychoClean.py", richiamate mediante il comando eval, ripulisce le variabili delle diverse colonne. Queste funzioni rimpiazzano i valori passati in input, i quali nella maggior parte dei casi sono stringhe, con valori numerici utili per il calcolo degli score, mediante l'ausilio di dizionari;
- Rinomina le colonne ove necessario;
- Rimuove gli spazi dagli "id_patient";
- Ripulisce le date;
- Numera i mesi in ordine crescente (questa operazione è necessaria per poter creare gli indici);
- Rinomina la colonna 'Clinical Center' a 'site_name' per uniformarla ai precedenti file Excel creati.

Non si sono rese necessarie operazioni di rimozione degli outlier in quanto siccome questi dati provengono da domande con risposte standard fatte mediante smartphone ai pazienti non presentano deviazioni dalle possibili risposte multiple.

I dati ripuliti sono stati inseriti nel file Excel “Psycho_clean_data2019.xlsx”, il quale ha sheet diverse per ognuno dei gruppi di domande.

Segue uno spaccato del codice con la funzione che effettua le principali operazioni di pulizia.

```
                                    .
                                    .
                                    .
def clean_and_save(data, writer, sheet):
    """Function that clean dates and save excel."""
    data = data.drop(columns=['id_visit', 'ordinal'])
    data['id_patient'] = data['id_patient'].map(
            lambda x: x.strip())
    data['date'] = clean_dates(data['date'])

    data = numerate_dates(data)

    # Rename Clinical Center in site_name
    data = data.rename(columns={'Clinical Center': 'site_name'})
    # Save data on excel
    data.to_excel(writer, sheet_name=sheet)
                                    .
                                    .
                                    .
```

4.4 Script per la pulizia dei dati relativi al lavoro dei pazienti

Anche in questo caso le risposte alla domanda relativa al lavoro dei pazienti è una domanda con tre risposte fisse (lavora, disoccupato, in pensione) di conseguenza non presenta outlier. Lo script (Script 2 Pulizia dati impiego) esegue le seguenti operazioni:

- Apre la sheet ‘share visit’ nel file relativo ai dati psicologici;

- Rinomina la colonna relativa allo stato lavorativo del paziente a 'unemployed';
- Converte i valori della precedente colonna a True se 'unemployed' o a False altrimenti;
- Ripulisce le date;
- Numera i mesi in ordine crescente (questa operazione è necessaria per poter creare gli indici);
- Rimuove gli spazi dagli "id_patient";
- Rinomina la colonna 'Clinic Center' a 'site_name' per uniformarla ai precedenti file Excel creati.

Salva il DataFrame ottenuto nel file Excel "Employ_clean_data2019.xlsx".

Segue il codice che effettua le operazioni descritte.

```
def EmploymentDataCleaner(file_psychological_data):
    """Method that clean data relative to patient employment
        status."""
    # Define clean data filename
    fpl = file_psychological_data
    fplclean = 'data/Employ_clean_data2019.xlsx'

    # Clean employment data
    datae = pd.read_excel(fpl,
                          sheet_name='ShareVisit')[['id_patient',
                                                    'date',
                'visita_which_is_your_current_employment_status',
                                                    'Clinical Center']
                                                     ]
    datae = datae.rename(columns={
      'visita_which_is_your_current_employment_status': 'unemployed'})
    datae['unemployed'] = datae['unemployed'].map(
        lambda x: True if x == 'Unemployed' else False)

    # Clean dates and enumerate months
    datae['date'] = clean_dates(datae['date'])
    datae = numerate_dates(datae)

    # Remove space before id_patients
    datae['id_patient'] = datae['id_patient'].map(lambda x: x.strip())

    # Rename Clinical Center in site_name
    datae = datae.rename(columns={'Clinical Center': 'site_name'})

    # Write data on file
    writer = pd.ExcelWriter(fplclean)
```

```
    datae.to_excel(writer, sheet_name='employment_status')
    writer.save()
```

Script 2 Pulizia dati impiego

4.5 Script per la pulizia dei dati Garmin

Questi sono tutti dati rilevati mediante un dispositivo elettronico(smartwatch) di conseguenza sono puliti. Lo script (Script 3 Pulizia dati Garmin) esegue le seguenti operazioni:

- Apre la sheet Garmin nel file relativo ai dati psicologici;
- Sostituisce il come della colonna relativo alla provenienza dei dati con 'site_name' per uniformarlo ai precedenti file Excel creati e rinomina 'sleep_in_second' a 'sleep';
- Ripulisce le date;
- Numera i mesi in ordine crescente (questa operazione è necessaria per poter creare gli indici);
- Rimuove gli spazi dagli "id_patient".

Salva il DataFrame ottenuto nel file "Garmin_clean_data2019.xlsx".

Segue il codice che effettua le operazioni descritte.

```
def GarminDataCleaner(file_psychological_data):
    """Method that clean data relative to garmin values."""
    # Define clean data filename
    fpl = file_psychological_data
    fplclean = 'data/Garmin_clean_data2019.xlsx'

    # Clean garmin data
    datag = pd.read_excel(fpl, sheet_name='Garmin')
    datag = datag.rename(columns={'company_name': 'site_name',
                                  'sleep_in_seconds': 'sleep'})

    # Clean dates and enumerate months
    datag['date'] = clean_dates(datag['date'])
    datag = numerate_dates(datag)

    # Remove space before id_patients
    datag['id_patient'] = datag['id_patient'].map(lambda x: x.strip())
```

```
    # Write data on file
    writer = pd.ExcelWriter(fplclean)
    datag.to_excel(writer, sheet_name='garmin')
    writer.save()
```

Script 3 Pulizia dati Garmin

4.6 Script per la pulizia degli score clinici

Questo script (Script 10 Pulizia score clinici) effettua la pulizia degli score utilizzati per effettuare le correlazioni.

Apre la sheet relativa alle visite, verifica la presenza del file relativo ai dati basici dei pazienti (se non lo trova invoca lo script che lo crea). Di seguito effettua le seguenti operazioni:

- Rimpiazza i valori '(null)', 999 o 'unknown' con il valore NumPy NaN;
- Unisce i DF;
- Rimuove gli spazi dai nomi delle colonne;
- Fa sì che i valori della colonna "Health_score" siano tra 0 e 10, effettuando le opportune divisioni;
- Rinomina le colonne relative alle cadute e alla perdita di equilibrio rispettivamente a 'Fall' e 'Loss_Balance';
- Converte i valori nelle colonne simil-booleane a valori booleani ('Yes' diventa True e 'No' diventa False);
- Verifica che le colonne numeriche siano tali convertendole;
- Rimuove la colonna "study_subject_id";

Infine, salva tutto nel file "Clinic_clean_scores2019.xlsx".

Segue uno spaccato di codice che effettua le principali operazioni di pulizia.

```
                                    .
                                    .
                                    .
                         # Remove invalid values
                        scores = scores.applymap(
    lambda x: np.nan if '(null)' in str(x)
                        or 'unknown' in str(x) or '999' in str(x)
                        else x)


# Merging data
datacs = pd.merge(subject, scores, on='study_subject_id')

# Remove space before column titles
datacs.columns = [c.lstrip() for c in datacs.columns]

# Convert values in Health Score in range 0-10
datacs['Health_score'] =
    datacs['Health_score'].astype(float).map(
            lambda x: x if 0 <= x <= 10 or x == np.nan
                         else np.round(x/10, 0))

# Rename columns with shorter names
datacs = datacs.rename(columns={
    'In_the_past_6_months_have_you_had_a_fall__label': 'Fall',

    'In_the_past_6_months_have_you_been_concerned_with_losing
    _your_balance_and_falling_while_doing_your_usual_daily_
    activities__label': 'Loss_Balance'})

# Replace fall data with True/False
datacs['Fall'] = datacs['Fall'].map(
    lambda x: True if x == 'Yes'
                    else False if x == 'No' else np.nan)
datacs['Loss_Balance'] = datacs['Loss_Balance'].map(
    lambda x: True if x == 'Yes'
                    else False if x == 'No' else np.nan)
datacs['EQ5D5L_score'] = datacs['EQ5D5L_score'].astype(float)
                                    .
                                    .
                                    .
```

4.7 Script numerazione date

Per quanto riguarda la numerazione delle date ho predisposto un semplice script che se ne occupa.

Lo script esegue le seguenti operazioni:

- Prende in input un DataFrame;
- Raggruppa gli elementi del DataFrame per paziente e applica a ogni sottoinsieme di dati relativi al paziente il sub-script che effettua le operazioni di numerazione delle date:
    - Il sub-script prende la colonna relativa alla data e la ordina in ordine crescente;
    - Raggruppa gli elementi in blocchi di 30 giorni (periodo dopo il quale il flusso di domande poste ai pazienti riparte);
    - Resetta l'indice e lo usa incrementato di uno per creare la colonna 'month';
    - Reimposta la colonna relativa alla data a stringa;
    - Ritorna il DataFrame del paziente modificato;
- Ritorna l'intero DataFrame modificato.

Segue il codice che effettua le operazioni descritte.

```
def enumerate_dates(df):
    """Method that add a sequential number for each quiz to
       patients."""
    def date_numerator(data):
        data['date'] = pd.to_datetime(data.date, dayfirst=True)
        data = data.sort_values('date')
        data = data.resample('30D',
               on='date').mean().reset_index().set_index('index')
        data['month'] = data.reset_index().index + 1
        data['date'] = data['date'].astype(str)
        return data

    df = df.reset_index().groupby('id_patient',
                                  'index').apply(
               date_numerator).reset_index().drop(columns='index')
    return df
```

Script 4 Numerazione date

Capitolo 5: Analisi e visualizzazione dei dati

In questo capitolo vado a descrivere come ho effettuato le operazioni di analisi dei dati con le quali ho potuto individuare gli errori presenti nei dati e attuare le procedure di pulizia degli stessi descritte nel precedente capitolo.

5.1 Analisi dei dati

L'operazione di analisi dei dati risulta molto complessa senza una rappresentazione grafica degli stessi, per questo motivo ho realizzato uno script che grazie a una web app consente di analizzare e filtrare i singoli parametri utilizzati per il calcolo degli indici, in modo da individuare eventuali problematiche presenti nei dati.

Oltre a questo script, all'interno dello script che si occupa della pulizia dei dati clinici (Script 8 Pulizia dati clinici) e relativi alle droghe ho inserito una porzione di codice (Script 5 Conteggio pazienti) che si occupa di contare i parametri presenti, assenti e con valori anomali dei vari pazienti, questi valori insieme ai conteggi dei parametri psicologici e Garmin sono stati usati per capire la mole di dati di progetto.

Per rappresentare i conteggi dei dati ho usato dei semplici e intuitivi istogrammi, realizzati mediante la libreria Seaborn, lo script che esegue queste funzionalità ha tre funzioni principali che creano tre file diversi, uno per gruppo di dati (Clinici, Psicologici, Garmin).

```
                                        .
                                        .
                                        .
# Create a DataFrame for values
data_count = datac.groupby(['event_name',
                            'site_name']).apply(
            lambda x: x.apply(
            lambda y: pd.Series(
                           {'GOOD': y[~y.isin([-1, -2])].count(),
                            'NOTGOOD': y[y == -2].count(),
                            'ABSENT': y[y == -1].count()})))
                                        .
                                        .
                                        .
```

Script 5 Conteggio pazienti

5.1.1 Visualizzazione dei dati

Per visualizzare i dati, ho creato una applicazione web in grado di mostrare i diversi parametri usati per la creazione degli indici in modo semplice e intuitivo. Questa web app è realizzata mediante uno script (Script 11 Web app dati) che fa uso della libreria Dash.

Questo script, come anche quello usato per la visualizzazione degli indici, offre delle funzionalità all'utente, le quali sono:

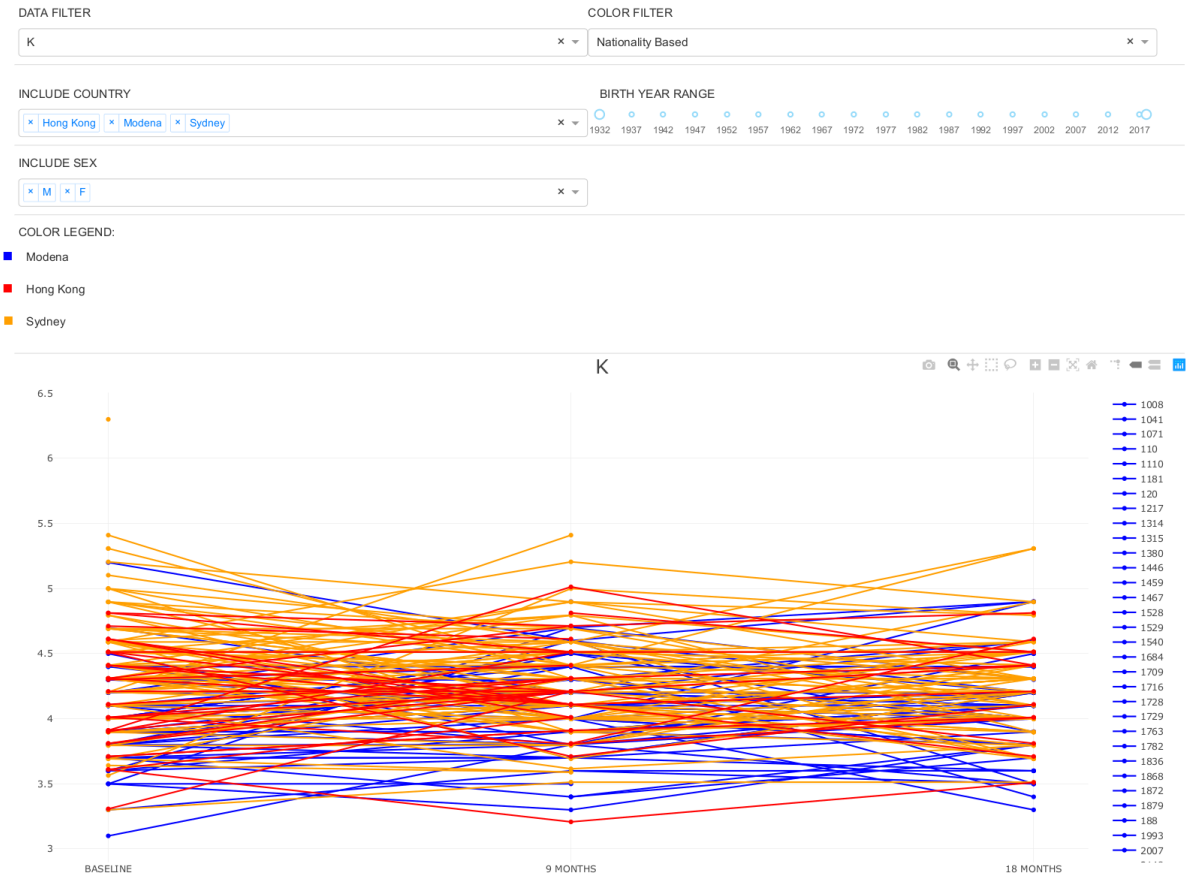
- Possibilità di scegliere il parametro da stampare;
- Possibilità di raggruppare per colore i dati, i dati possono essere visualizzati con colori diversi in base:
    - Al paziente, in questo caso i pazienti hanno colori diversi;
    - Alla provenienza, in questo caso i pazienti hanno colori basati sulla loro provenienza (Blu – Modena, Giallo – Sydney, Rosso – Hong Kong);
    - All'età del paziente, in questo caso il colore delle linee rappresentanti i pazienti è via via più scuro più i pazienti sono anziani;
- Possibilità di filtrare il sesso dei pazienti;
- Possibilità di filtrare la provenienza dei pazienti;
- Possibilità di filtrare il range di età dei pazienti.

La scelta di queste funzionalità viene fatta mediante dei menù nella parte superiore della pagina web.

Scelta la colorazione viene inserita la legenda dei colori al di sotto dei menù.

Al di sotto della legenda vi è il grafico vero e proprio, il quale rispecchia i filtraggi fatti. Essendo un grafico creato con la libreria Plotly, è interattivo, ossia si può selezionare un determinato paziente dalla lista alla destra di esso e si possono analizzare i valori dei diversi punti del grafico puntandoli con il mouse.

Per completezza al di sotto del grafico vi è la tabella, la quale rispecchia i filtraggi effettuati e aggiunge alcune statistiche sui dati, come deviazioni standard, valore medio, variazioni tra i diversi periodi e il conteggio dei dati.



id_patient	BASELINE	9 MONTHS	18 MONTHS	BASELINE COUNT	9 MONTHS COUNT	18 MONTHS COUNT	stdv	mean	delta 0-9	delta 9-18	delta 0-18
Hong Kong	4.16	4.23	4.22	29	30	14	0.04	4.2	0.07	-0.01	0.06
Modena	3.98	4.08	4.04	54	100	61	0.05	4.03	0.1	-0.04	0.06
Sydney	4.41	4.3	4.26	96	86	65	0.08	4.32	-0.11	-0.04	-0.15
MEAN	4.24	4.19	4.16								
COUNT	179	216	140								

Figura 5-1 Web app dati

Lo script che esegue la web app è contenuto nel file “ClinicaTrend3.0.py” ed esegue le seguenti operazioni:

- Carica le stylesheet necessarie per la web app;
- Crea l’oggetto Dash che rappresenta la web app;
- Carica i dati clinici, psicologici, relativi alla situazione lavorativa, del dispositivo Garmin e li inserisce in DataFrame;
- Per effettuare le operazioni di filtraggio, crea una serie con le provenienze dei pazienti, una con gli anni di nascita dei pazienti e una con il sesso dei pazienti;
- Crea un dizionario con la palette dei colori usata per la colorazione basata sulla provenienza;
- Per creare il dizionario con la palette colori usata per la colorazione basata sull’anno di nascita dei pazienti crea dieci range di età il più possibile equipotenti, ad ognuno di essi viene associato un colore della scala di verde, più è scuro più il paziente è anziano, più è chiaro più il paziente è giovane;
- Definisce una lista con i parametri per i quali non viene creato un grafico;
- Per i parametri da plottare va a creare delle pivot-table, inserendole in un dizionario per rendere facilmente accessibili;
- Crea il layout che avrà la web app in una codifica simil html implementata con la libreria “dash_html_components” (implementa i vari tag html) e “dash_core_components” (definisce le componenti Dash inseribili nella pagina html);
- Definisce la funzione “update_graph(select_data, select_coloring, include_countries, age_range, include_sex)”, la quale viene invocata mediante dei callback azionati dalla modifica di uno dei filtri;

  Questa funzione esegue le seguenti operazioni:

  - Seleziona dal dizionario la pivot-table da stampare in base alla stringa ritornata dal calback nella variabile “select_data”;
  - Crea gli oggetti che conterranno il grafico, la legenda e la tabella;

- Crea il DataFrame che conterrà la tabella che verrà inserita al di sotto del grafico;
- Per ogni riga della pivot-table (paziente-eventi/mesi):
    - Verifica che il paziente abbia una provenienza tra quelle filtrate (la città del paziente deve essere nella lista “include_countries”);
    - Verifica che l’anno di nascita del paziente sia nel range definito dal filtro (l’anno di nascita deve essere nel range definito dalla lista “age_range”);
    - Verifica che il sesso del paziente sia tra quelli definiti dal filtro (il sesso del paziente deve essere nella lista “include_sex”);
    - Crea l’hover-text per i diversi punti della linea che descrive l’andamento del parametro selezionato nel tempo;
    - Se la colorazione selezionata è basata sulla provenienza del paziente, va a scegliere il colore in base alla provenienza del paziente e aggiunge una riga contenente la riga considerata e la provenienza del paziente al DataFrame usato per la creazione della tabella;
    - Se la colorazione selezionata è basata sull’anno di nascita del paziente, va a scegliere il colore in base alla provenienza del paziente e aggiunge una riga contenente la riga considerata e l’anno di nascita del paziente al DataFrame usato per la creazione della tabella;
    - Se la colorazione è diversa per ogni paziente, viene solamente aggiunta la riga considerata al DataFrame usato per creare la tabella;
    - Viene aggiunta la linea relativa alla riga al grafico.
- Viene definito il layout del grafico;
- Viene creata la riga con i valori medi per ogni evento/mese;

- Viene creata la riga con i conteggi dei valori per ogni evento/mese;
- Se la colorazione è basata sulla provenienza o sull'età dei pazienti, viene inserita la legenda relativa alle colorazioni, creata con oggetti html, nella figura legenda;
- Se i dati necessari per la creazione della tabella sono presenti, viene creata la colonna con i conteggi degli eventi/mesi con valori validi per ogni utente nel caso sia selezionata la modalità di visualizzazione basata sul paziente, per ogni nazione nel caso sia selezionata la colorazione basata sulla provenienza oppure per ogni range di età nel caso sia selezionata la colorazione basata sull'età dei pazienti;
- Vengono aggiunte le colonne contenti i valori medi e le deviazioni standard per ogni riga;
- Per ogni riga vengono aggiunte tre colonne contenenti le variazioni tra i 3 diversi periodi (0-9 mesi, 9-18 mesi e 0-18 mesi);
- Vengono inserite le righe precedentemente create contenenti i conteggi e i valori medi;
- Nella figura tabella definita in precedenza, viene inserita la tabella Dash contenente la tabella precedentemente creata;
- Vengono ritornate: la figura contenente il grafico, la figura contenente la legenda e la figura contenente la tabella, le quali andranno ad aggiornare quelle presenti nella web app.

5.1.2 Funzione per il conteggio dei dati clinici

Questa funzione (Script 12 Conteggio dati clinici) si occupa della creazione di tre istogrammi che rappresentano il conteggio dei dati clinici nei diversi eventi.

Per la creazione di questi istogrammi legge il file contenente i conteggi dei dati, creato durante la fase di pulizia degli stessi e crea quelli che vengono chiamati "stacked plots",

ossia plot sovrapposti, nei quali le barre che costituiscono l'istogramma sono suddivise per colore in base a determinate caratteristiche.

In questo caso i colori sono tre:

- *Verde*: Utilizzato per rappresentare i valori accettabili;
- *Blu*: Utilizzato per rappresentare i valori assenti;
- *Rosso*: Utilizzato per rappresentare i valori ritenuti non accettabili durante la fase di pulizia dei dati.

Questo programma va a produrre un file il quale è salvato all'interno della cartella plots.

Segue uno spaccato dell'immagine prodotta.

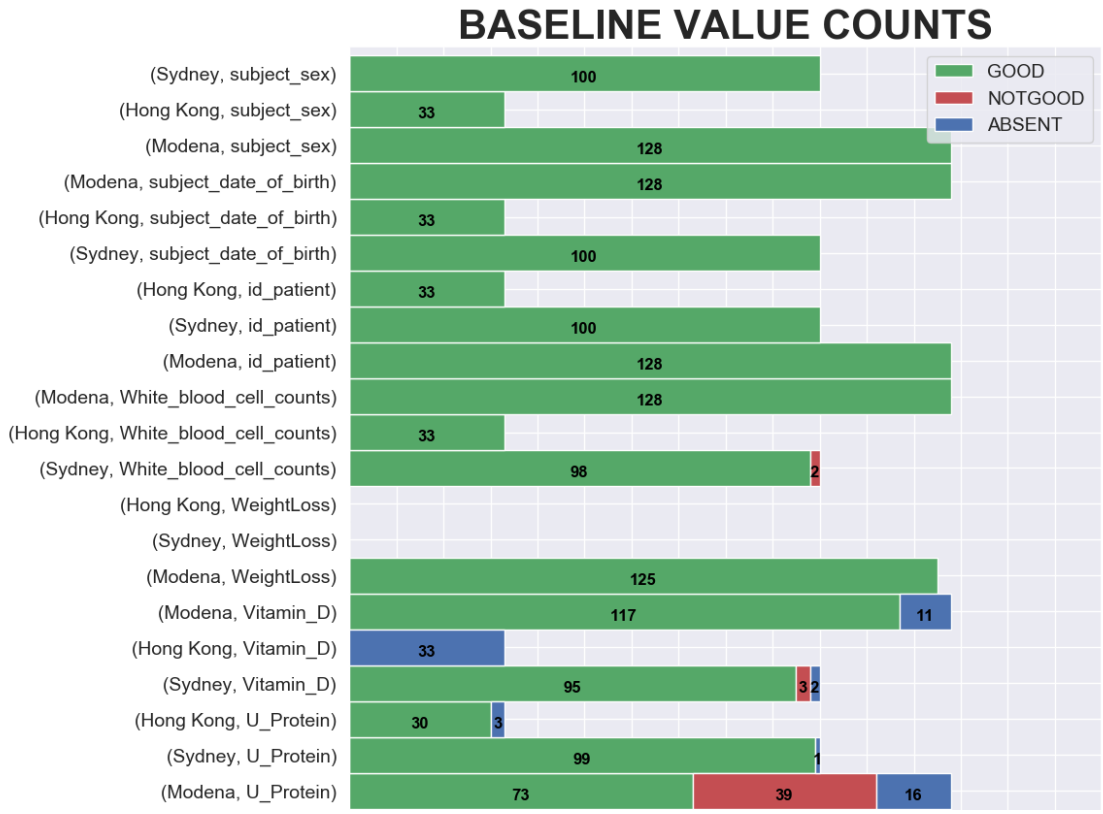


Figura 5-2 Conteggio dati clinici

Come si può vedere dall'immagine precedente i dati riguardanti la proteina U, provenienti dal centro clinico di Modena presentano un gran numero di errori. Da un confronto con i medici è emerso che alcuni valori, tra i quali la proteina U, non erano stati considerati come utili inizialmente e pertanto in molti casi non registrati nel database.

5.2 Funzione per il conteggio dei dati psicologici

Questa funzione (Script 13 Conteggio dati psicologici) si occupa della creazione di un numero istogrammi pari al numero di sheet del file Excel contenente i dati psicologici.

Per la creazione di questi istogrammi legge il file contenente i dati psicologici e quello relativo all'impiego svolto dai pazienti, andando a contare per ogni sheet (gruppo di questionari) quanti pazienti sono presenti in ogni mese. Di seguito crea un file con tanti subplot quanti sono i grafici, in ognuno di questi subplot sulle righe è presente il valore numerico relativo al conteggio, sulle colonne vi sono i mesi mentre le provenienze dei pazienti sono rappresentate mediante i colori:

- *Blu*: Modena;
- *Giallo*: Sydney;
- *Rosso*: Hong Kong.

La realizzazione dei subplot è affidata a una funzione che verrà descritta successivamente.

Segue uno spaccato dei conteggi dei dati psicologici per i primi nove mesi.

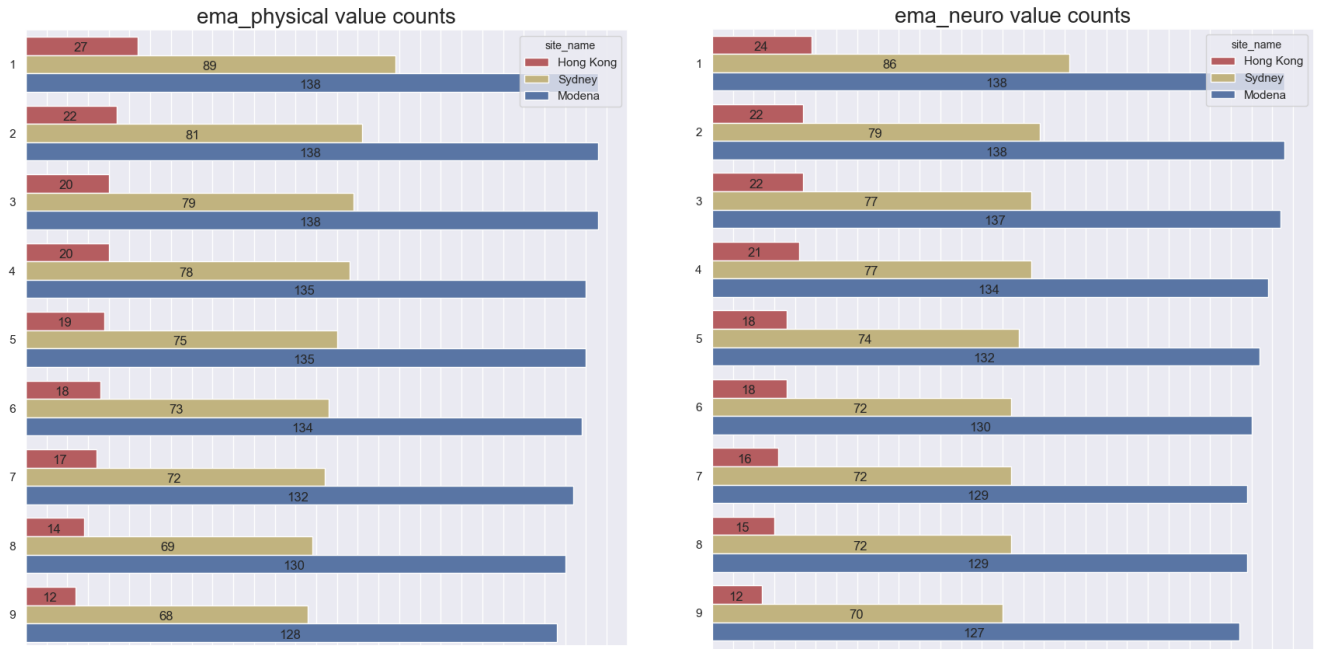


Figura 5-3 Conteggio dati psicologici

5.3 Funzione conteggio dati Garmin

Questa funzione (Script 6 Conteggio dati Garmin) si occupa della creazione di un istogramma contenente il conteggio dei dati relativi al dispositivo Garmin.

Per la creazione di questo istogramma legge il file contenente i dati Garmin contando quanti pazienti sono presenti in ogni mese. Di seguito crea un file con grafico nelle cui righe è presente il valore numerico relativo al conteggio, sulle colonne vi sono i mesi mentre le provenienze dei pazienti sono rappresentate mediante i colori:

- *Blu*: Modena;
- *Giallo*: Sydney;
- *Rosso*: Hong Kong.

La realizzazione del plot è affidata a una funzione che verrà descritta successivamente.

Segue il codice per la realizzazione di questo plot.

```
def get_count_garmin_data():
    """Script that create a file with an histogram relative to patient
        count in each month."""
    file_garmin_data = 'data/Garmin_clean_data2019.xlsx'
    file_pdf_output = 'plots/Garmin_Counts2019.png'

    # Set figure size and figure parameters
    fig = plt.figure(figsize=(10, 30))

    # Set style
    sns.set(style='darkgrid')

    # Create a subplot for garmin data
    ax = fig.add_subplot(1, 1, 1)
    prepare_and_plot_data(file_garmin_data, 'garmin', ax)

    # Save plot on file
    plt.savefig(file_pdf_output, bbox_inches='tight')
```

Script 6 Conteggio dati Garmin

Segue lo spaccato dei primi nove mesi del grafico prodotto.

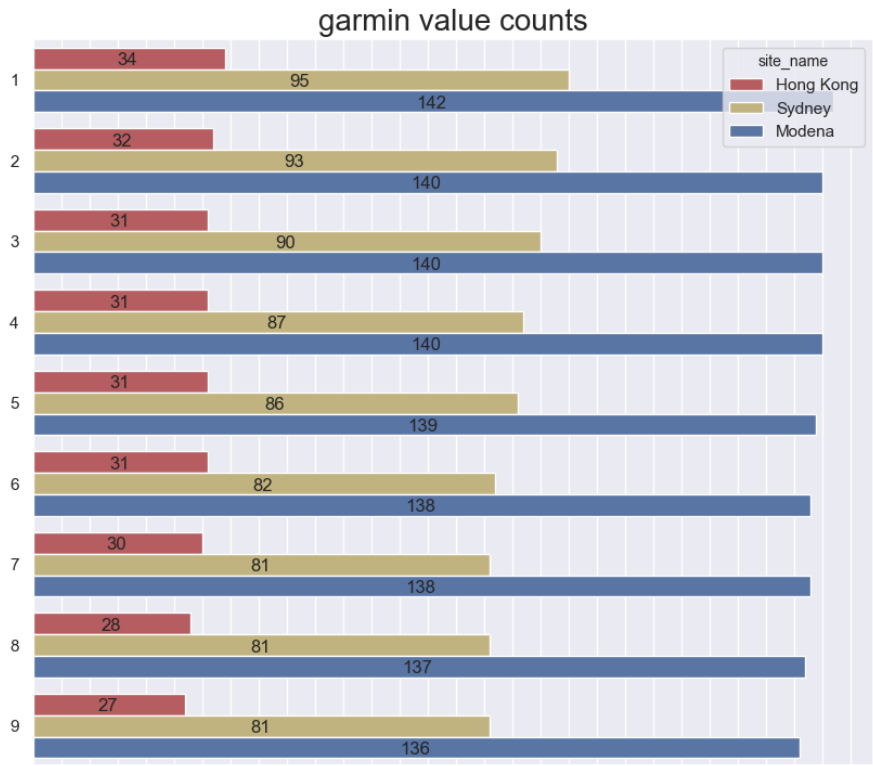


Figura 5-4 Conteggio dati Garmin

5.4 Funzione per la creazione dei plot

Questa funzione si occupa di plottare il grafico contenete i conteggi dei pazienti per ogni mese, per farlo:

- Apre il file Excel passato come parametro nella sheet passata come parametro;
- Aggiunge la colonna “site_name” nei DataFrame nel quale non è presente;
- Conta i pazienti di ogni centro di ricerca in ogni mese;
- Ordina i conteggi per mese crescente;
- Setta la palette dei colori;

- Crea l'istogramma all'interno degli assi (subplot) passati come parametro utilizzando la libreria Seaborn;
- Aggiunge le annotazioni ad ogni barra in modo da consentire una lettura veloce del valore rappresentato;
- Setta i valori dei tick, il nome delle ascisse e delle ordinate e il titolo del subplot.

Segue il codice che effettua le operazioni descritte.

```
def prepare_and_plot_data(excel_file, sheet, ax):
    """Script that plot patient counts in a subplot."""
    # Read an excel file
    data = pd.read_excel(excel_file, sheet_name=sheet, index_col=0)
    data.date = pd.to_datetime(data.date, dayfirst=True)

    # Add site name for certain plots
    if 'site_name' not in data.columns.tolist():
        data = data.merge(pd.read_excel(patient_data,
                          index_col=0)[['id_patient', 'site_name']],
                          on='id_patient')

    # Count patient in each months
    counts = data.groupby(['site_name',
                           'month']).count().reset_index()[
                               ['id_patient', 'month', 'site_name']]
    counts = counts.sort_values(by='month', ascending=True)

    # Plot values on histogram
    colors = {'Modena': 'b',
              'Sydney': 'y',
              'Hong Kong': 'r'}
    splot = sns.barplot(data=counts,
                        orient='horizontal',
                        x='id_patient',
                        y='month',
                        hue='site_name',
                        palette=colors,
                        ax=ax)
    # Add annotations on bars
    for p in splot.patches:
        if not str(p.get_width()) == 'nan':
            splot.annotate(int(p.get_width()),
                           (p.get_width()/2 -
                               len(str(int(p.get_width()))),
                            p.get_y() + p.get_height() - 0.05))
    # Set tick, labels and title
    ax.set_xlabel('COUNT', size=14)
    ax.set_ylabel('MONTHS', size=14)
    ax.set(xticks=(np.arange(0, 150, 5)))

    # Set subplot title
    plt.title(sheet + ' value counts', fontsize=20)
```

Script 7 Prepara e stampa i conteggi

5.5 Conclusioni riguardanti l'analisi dei dati

Grazie a queste operazioni per il conteggio e la visualizzazione dei dati si è potuto osservare l'assenza di molti dei dati, queste mancanze per quanto riguarda i pazienti di Modena sono state in gran parte risolte estraendo i dati dal database dell'ospedale, al contrario per questioni di tempo non si è potuto intervenire sui dati dei pazienti di Hong Kong e Sydney per i quali sono presenti delle lacune che portano all'assenza dei valori degli indici in alcuni periodi.

Per ovviare al problema della mancanza di dati, si è scelto di abbassare la soglia di precisione per il calcolo degli indici dall'ottanta percento al settanta percento di valori validi (diversi da NumPy NaN) dei deficit necessari per il calcolo degli stessi.

Capitolo 6: Calcolo degli indici

In questo capitolo vado a descrivere come ho effettuato il calcolo degli indici basandomi sulle soglie fornite nei requisiti.

Per la creazione degli indici ho predisposto due classi, uno per l'indice di fragilità e uno per l'indice di capacità intrinseca.

6.1 Classe per il calcolo del Frailty Index NO MM

Questa classe (Script 14 Classe per il calcolo del FI) si occupa del calcolo dell'indice di fragilità senza multi-morbosità, restituendo un file il valore dell'indice per ogni paziente in ogni evento.

Una volta creato l'oggetto della classe vengono caricati tutti i file Excel necessari, per il calcolo di questo indice sono necessari:

- Il file con i dati clinici puliti ("Clinic_clean_data2019.xlsx");
- Il file con i dati psicologici ("Psycho_clean_data2019.xlsx").

Le operazioni effettuate successivamente sono:

- Lettura del file contenente i dati clinici;
- Lettura delle sheet contenenti i dati psicologici necessari per il calcolo del FI NOMM (le sheet e i parametri necessari sono definiti nel dizionario "fi_NOMM_psycho" nel file "parameterFINOMMIndex.py");
- Lettura del file contenente i dati relativi ai farmaci assunti dai pazienti.

I dati letti sono inseriti in opportuni DataFrame.

6.1.1 Metodo per il calcolo del Frailty Index

Questo metodo al suo interno ha una funzione che viene applicata a tutte le righe (paziente-evento).

Tale funzione esegue le seguenti operazioni:

- Crea due variabili usate per lo score e per il conteggio dei valori validi (non NumPy NaN);
- Prende il valore relativo al sesso del paziente (necessario per il calcolo di alcuni punteggi);
- Per ognuno dei parametri necessari per il calcolo degli indici (lista "fi_NOMM_param" nel file "parameterFINOMMIndex.py"):
    - Preleva il valore corrispondente dalla riga;
    - Se il valore non è NumPy NaN invoca la funzione corrispondente nel file "parameterFINOMMIndex.py" passando il valore e se necessario (se il parametro è nella lista "gender_param") passa anche il valore corrispondente al sesso del paziente.

      Il valore di ritorno della funzione viene sommato alla variabile che gestisce gli score e viene incrementata la variabile che gestisce i conteggi dei valori validi;
- Viene prelevato l'id del paziente e l'evento;
- Viene estratto il valore in mesi relativo all'evento;
- Si accede al DataFrame relativo all'assunzione di farmaci e si conta quanti farmaci il paziente sta assumendo, se il paziente assume più farmaci di quelli previsti dalla soglia (variabile polipharmacyTreshold nel file "parameterFINOMMIndex.py") allora si incrementa lo score, il conteggio dei parametri in questo caso viene sempre incrementato;
- Per i parametri psicologici necessari per il calcolo del FI NOMM:
    - Viene prelevato il valore relativo al paziente-mese dalla variabile corrispondente al parametro;

    - Se il valore non è NumPy NaN viene calcolato lo score (invocando la funzione corrispondente nel file “parameterFINOMMIndex.py”), aggiunto alla variabile che gestisce gli score e incrementata la variabile che conta i parametri validi;
- Se il numero dei parametri validi è maggiore del numero minimo di parametri per cui il valore del FI NOMM è valido (“fi_NOMM_min_values” nel file “parameterFINOMMIndex.py”, attualmente pari al 70 % del numero di parametri totali utilizzati per il calcolo del FI NOMM) allora viene ritornato il valore del FI NOMM per paziente-evento, pari a:

  Valore dello score / Numero di parametri validi usati

  Altrimenti viene ritornato NumPy NaN.

I valori di FI NOMM vengono salvati nella colonna ‘fi_val’ del DataFrame originale.

Viene creata una pivot-table con indice il paziente e colonne gli eventi, salvata nella variabile di classe fi_NOMM e la quale è ritornata.

6.1.2 Metodo per il salvataggio del FI

Questo metodo semplicemente salva nel file Excel “FI_NOMM_index2019.xlsx” la pivot-table prodotta dal metodo precedente.

6.2 Classe per il calcolo del Intrinsic Capacity Index

Questa classe (Script 15 Classe per il calcolo del ICI) si occupa del calcolo dell'indice di capacità intrinseca, restituendo un file il valore dell'indice per ogni paziente in ogni mese.

Una volta creato l'oggetto della classe vengono caricati tutti i file Excel necessari, per il calcolo di questo indice sono necessari:

- Il file con i dati relativi ai pazienti ("PatientBasicData2019.xlsx");
- Il file con i dati relativi allo stato lavorativo dei pazienti ("Employ_clean_data2019.xlsx");
- Il file con i dati psicologici ("Psycho_clean_data2019.xlsx");
- Il file con i dati Garmin ("Garmin_clean_data2019.xlsx").

Le operazioni effettuate successivamente sono:

- Lettura dei file contenente i dati relativi ai pazienti;
- Lettura del file contenente i dati relativi allo stato lavorativo dei pazienti;
- Lettura del file contenente i dati di Garmin;
- Lettura delle sheet contenenti i dati psicologici necessari per il calcolo del IC (le sheet e i parametri necessari sono definiti nel dizionario "ic_parameters" nel file "parameterICIndex.py").

I dati letti sono inseriti in opportuni DataFrame.

6.2.1 Metodo per il calcolo del Intrinsic Capacity Index

Questo metodo al suo interno ha una funzione che viene applicata a tutti i pazienti presenti nel DataFrame che contiene i dati dei pazienti, per i mesi che vanno dal primo fino a NOM (variabile presente nel file "parameterICIndex.py"). I valori di IC ritornati dalla funzione vengono inseriti in una serie.

Tale funzione che prende in ingresso il paziente e il mese, esegue le seguenti operazioni:

- Crea due variabili usate per lo score e per il conteggio dei valori validi (non NumPy NaN);
- Nel DataFrame contenente i valori relativi all'impiego dei pazienti preleva i valori relativi al paziente in questione, se non trova il valore di impiego nel mese passato alla funzione (molto probabile visto che la domanda riguardante lo stato lavorativo del paziente non viene posta tutti i mesi) va a ritroso nei mesi precedenti finché non la trova o non arriva al primo, se trova un valore valido lo aggiunge allo score e incrementa il contatore dei parametri validi utilizzati, altrimenti prosegue;
- Per i parametri psicologici necessari per il calcolo del IC (presenti nel dizionario "ic_parameters" nel file "parameterICIndex.py"):
    - Viene prelevato il valore relativo al paziente-mese dalla variabile corrispondente al parametro;
    - Se il valore non è NumPy NaN viene calcolato lo score (invocando la funzione corrispondente nel file "parameterICIndex.py"), viene aggiunto lo score alla variabile che gestisce gli score e incrementata la variabile che conta i parametri validi.

  L'invocazione di queste funzioni viene fatta o chiamando direttamente la funzione (per i parametri riguardanti "ema_mood" o "ema_eat") oppure mediante il metodo eval che invoca la funzione con il nome del parametro.

- Vengono prelevati i valori relativi al paziente-mese dal DataFrame contenente i dati Garmin;
- Viene fatta la media dei valori del mese (i dati Garmin sono prelevati settimanalmente);
- Per i parametri Garmin necessari per il calcolo del IC (presenti nel dizionario "ic_garmin" nel file "parameterICIndex.py"):

    - Se il valore corrispondente al parametro non è NumPy NaN, viene calcolato lo score (invocando la funzione corrispondente nel file "parameterICIndex.py"), viene aggiunto lo score alla variabile che gestisce gli score e incrementata la variabile che conta i parametri validi;
- Se il numero dei parametri validi è maggiore del numero minimo di parametri per cui il valore del IC è valido ("ic_min_values" nel file "parameterICIndex.py", attualmente pari al 70 % del numero di parametri totali utilizzati per il calcolo del IC) allora viene ritornato il valore del IC per paziente-mese, pari a:

  Valore dello score / Numero di parametri validi usati

  Altrimenti viene ritornato NumPy NaN.

Una volta che la serie è completa viene aggiunta alla variabile di classe contenente il DataFrame con tutti i valori mensili dei diversi pazienti.

Il DataFrame finale avrà già l'aspetto di una pivot-table con gli id dei pazienti come indice di riga e i mesi come colonne.

Al termine del metodo questo DataFrame viene ritornato.

6.2.2 Metodo per il salvataggio del ICI

Questo metodo semplicemente salva nel file Excel "IC_index2019.xlsx" la pivot-table prodotta dal metodo precedente.

6.3 Visualizzazione degli indici

La visualizzazione degli indici è fatta in due versioni, una dinamica e una statica.

6.3.1 Visualizzazione dinamica degli indici

La visualizzazione dinamica degli indici è realizzata con uno script del tutto simile a quello usato per la visualizzazione dinamica dei dati, con l'unica differenza che al posto dei dati vi sono gli indici.

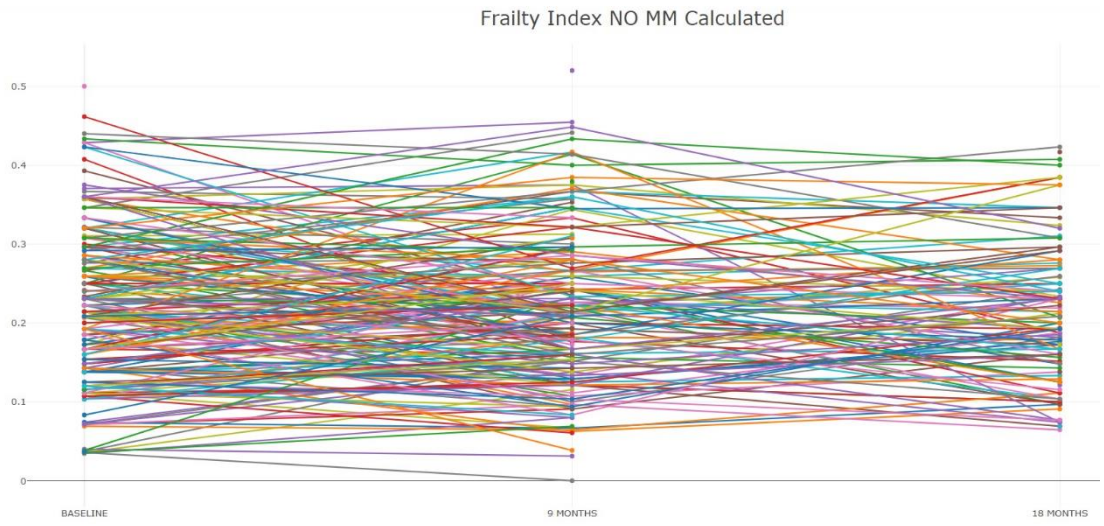


Figura 6-1 FI NO MM Plotly

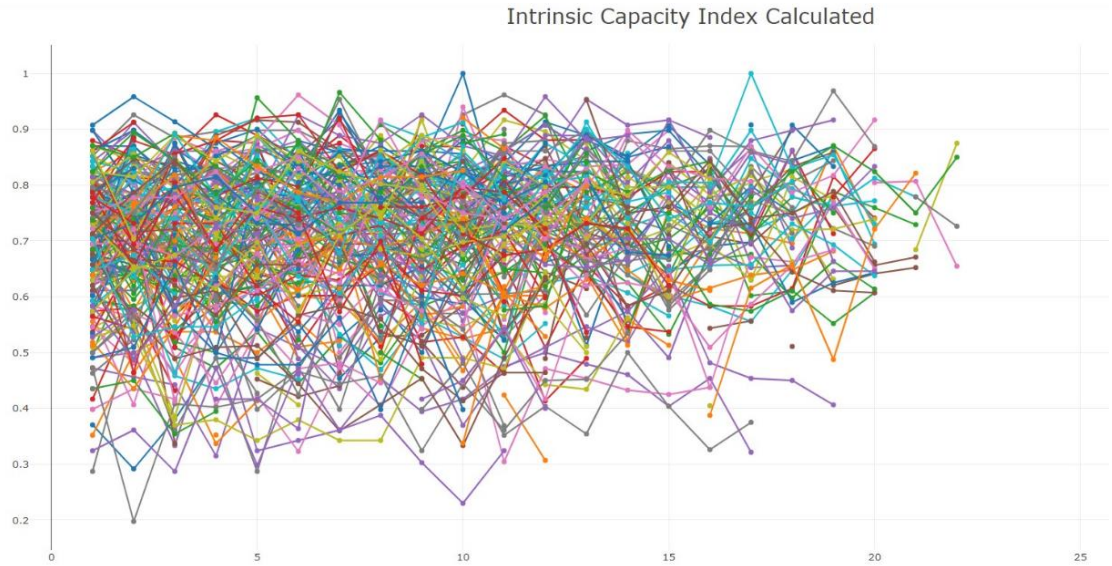


Figura 6-2 IC Plotly

6.3.2 Visualizzazione statica degli indici

La visualizzazione statica degli indici è realizzata mediante lo script (Script 16 Heatmap degli indici) “indexesHeatmapImage.py” il quale va a creare un’immagine (.png) con tre heatmap, una per ogni centro di ricerca, rappresentanti gli indici per ogni paziente in colonne sovrapposte, in modo da poter analizzare l’andamento di un indice rispetto all’altro.

Questo script esegue le seguenti operazioni:

- Definisce i centri di ricerca da plottare;
- Definisce gli indici da plottare;
- Definisce i mesi da plottare (1-18);
- Va a definire i colori delle per le heatmap e per la legenda delle heatmap;
- Carica gli indici in opportuni DataFrame;
- Dal file con i dati dei pazienti preleva una serie con le provenienze dei pazienti;
- Va a convertire i valori del FI NO MM in valori mensili assegnando i valori di “BASELINE” ai mesi nel range 1-8, i valori di “9 MONTHS” ai mesi nel range 9-17 e i valori di “18 MONTHS” ai mesi nel range 18-26;
- Per ridurre i tempi di creazione del grafico, i valori di FI NO MM sono stati invertiti al corrispondente negativo, così da poter applicare il metodo per la creazione della heatmap con un’unica colormap a tutto il DataFrame anziché riga per riga;
- Crea DataFrame diversi per ogni centro di ricerca;
- Per ogni riga del DataFrame con le provenienze dei pazienti, se la provenienza è tra quelle da inserire nel grafico:
    - Per ogni indice nella lista degli indici da plottare, preleva il paziente corrispondente dal DataFrame contenente tale indice;
    - Aggiunge l’indice all’id del paziente;
    - Se la riga è vuota crea una riga con valori NumPy NaN in ogni colonna;

    - Aggiunge questa riga al DataFrame relativo alla provenienza del paziente;
- Per far sì che le tre heatmap avessero caselle della stessa dimensione ho calcolato la percentuale di sazio da assegnare a ciascuna heatmap contando la percentuale di righe di ciascun DataFrame “nazionale” rispetto al totale delle righe di tutti i DataFrame “nazionali”;
- Ho impostato lo stile Seaborn per il grafico;
- Ho creato le colormap basandomi sulle palette definite in precedenza;
- Ho creato la figura con un numero di subplot pari al numero di centri di ricerca coinvolti più uno, per la legenda, impostando le dimensioni;
- Ho creato la legenda, suddividendo il suo subplot in subplot più piccoli in ognuno dei quali ho inserito una colorbar con i colori di uno degli indici di un centro di ricerca;
- Per i centri di ricerca coinvolti ho plottato una heatmap, con l’apposita colormap precedentemente creata e il valore assoluto del valore corrispondente in ognuna delle caselle (il valore assoluto serve per avere tutti valori positivi in quanto per poter usare una sola colormap i valori di FI NO MM erano stati invertiti);
- Dopo aver spostato in alto l’asse delle ascisse (mesi) e aver dato un titolo agli assi, i plot sono salvati nel file “IndexesHeatmap.png”.

Ecco uno spaccato del risultato (le linee verdi rappresentano i punti in cui sono stati effettuati i tagli).

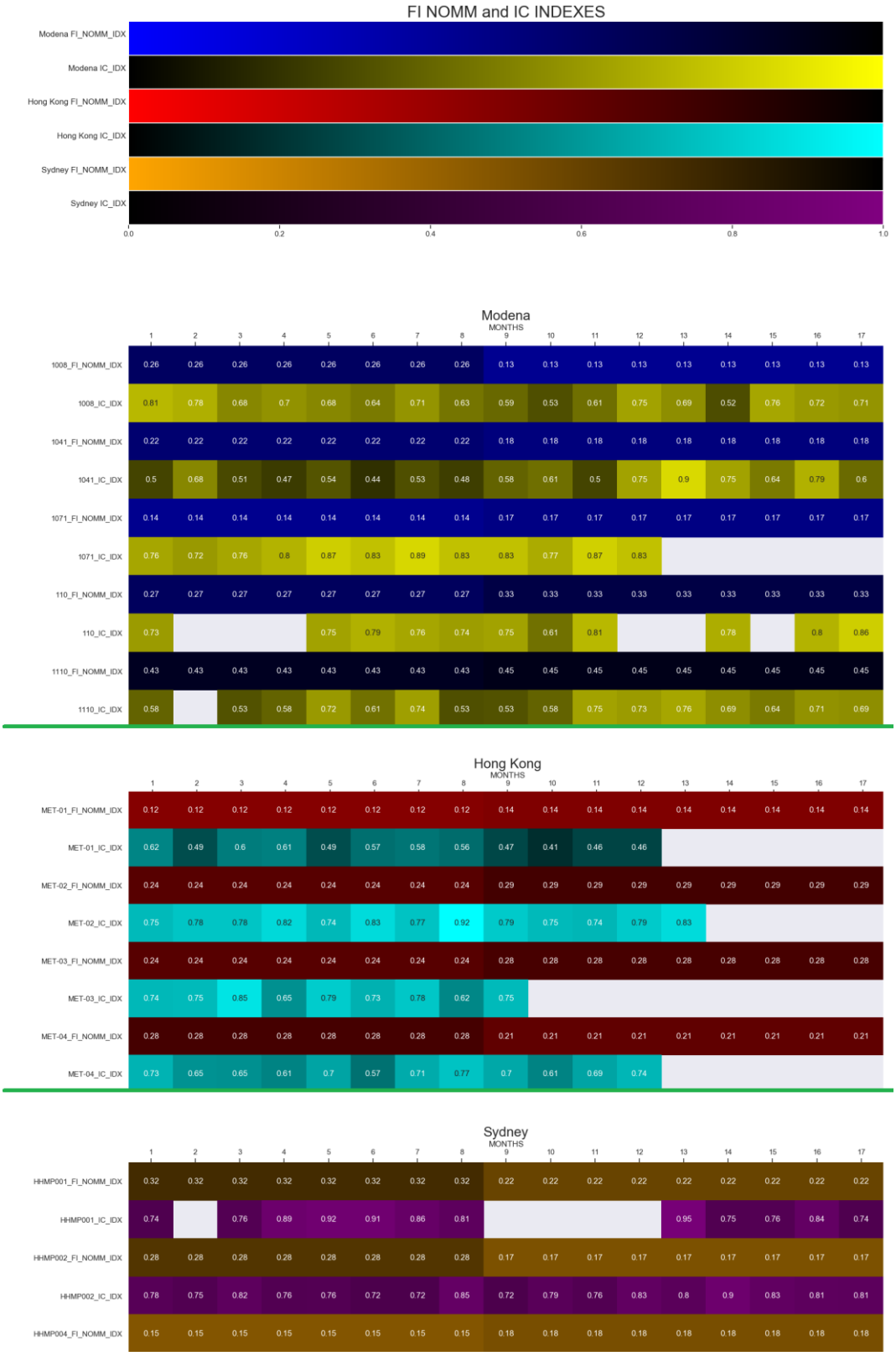


Figura 6-3 Heatmap indici Seaborn

6.4 Osservazioni sugli indici

Come è possibile osservare nell'immagine rappresentante l'heatmap degli indici, vi sono delle lacune, la maggior parte delle quali si concentrano nei valori dell'indice di capacità intrinseca; queste lacune sono principalmente dovute alla mancata risposta ai questionari da parte dei pazienti e per tale motivo questi valori non sono reperibili.

Capitolo 7: Correlazione dei dati

In questo capitolo vado a descrivere le operazioni di correlazione degli indici, questa operazione è necessaria per capire se gli indici forniscono una informazione utile a comprendere lo stato di salute dei pazienti.

7.1 Classe per la correlazione degli indici

Per dare un significato agli indici e capire come questi cambiano al cambiare degli score ho realizzato una classe (Script 17 Classe per le correlazioni) che correla gli indici tra di loro e con gli score, utilizzando l'indice di correlazione di Spearman e Pearson, implementati con la libreria SciPy.

La classe che effettua le correlazioni è la classe "CorrelateS2I", contenuta nel file "CorrelateScoreToIndex.py". Creata un'istanza di questa classe, vengono effettuate le seguenti operazioni:

- Vengono caricati i file Excel contenenti gli indici e gli score clinici in tre DataFrame: uno per il FI NO MM, uno per l'IC e uno per gli score;
- Viene definito il file Excel di output;
- Viene creato il DataFrame per le correlazioni;
- Siccome i valori di IC sono mensili per le correlazioni si è scelto di raggrupparli per rappresentare i tre periodi (BASELINE, 9 MONTHS, 18 MONTHS) per farlo ho preso come valore di BASELINE il valore al mese 1, come valore a 9 MONTHS la media dei valori nel range di mesi da 2 a 9,

come valore a 18 MONTHS la media dei valori nel range di mesi da 10 a 18;

- Per ogni metodo di correlazione (pearson, spearman), ogni indice (FI NO MM, IC), per ogni score (Fall, Loss Balance, EQ5D5L, Health, KATZ, IADL), per ogni evento (BASELINE, 9 MONTHS, 18 MONTHS), crea un DataFrame con le due colonne. Mediante i metodi SciPy, correla le due colonne del DataFrame e inserisci il risultato delle correlazioni e il p-value in un DataFrame delle correlazioni;
- Per ogni metodo, per ogni evento crea una riga del DataFrame delle correlazioni per la correlazione dei due indici e per il p-value;
- Salva il DataFrame creato nel file Excel "scoreIndexexCorrelations2019.xlsx".

7.1.1 Metodo per la stampa con Seaborn

Metodo utilizzato per plottare un'immagine statica (Figura 7-1 Heatmap correlazioni) delle correlazioni.

Questo metodo effettua le seguenti operazioni:

- Imposta lo stile Seaborn;
- Crea una figura con un numero di subplot pari ai metodi utilizzati e ne imposta le dimensioni;
- Per ogni metodo utilizzato per creare le correlazioni, crea due pivot-table del DataFrame con gli eventi come colonne e i titoli delle diverse correlazioni come righe, in una vi inserisce i valori delle correlazioni e nell'altra i valori dei p-value;
- Arrotonda i valori di p-value alla 3 cifra decimale;
- Crea la tabella con le annotazioni per ogni cella della heatmap nel formato "valore correlazione (valore p-value)";
- Crea una colormap;

- Crea una heatmap usando l'opportuna funzione della libreria Seaborn;
- Salva l'immagine nel file "scoresIndexesCorrelations2019.png".

7.1.2 Metodo per la stampa con Plotly

Metodo che va a creare una pagina html con un grafico a heatmap interattivo, rappresentante le correlazioni.

Questo metodo esegue le seguenti operazioni:

- Crea una figura con tanti subplot quanti sono i metodi utilizzati per le correlazioni;
- Imposta il layout della figura;
- Definisce una colormap centrandola sullo 0;
- Per ogni metodo di correlazione:
    - Crea una pivot-table con gli eventi come colonne, i titoli delle diverse correlazioni come righe e i valori delle correlazioni nelle celle;
    - Crea l'hover-text da assegnare a ogni cella della heatmap;
    - Aggiunge la heatmap alla figura;
- Salva la figura creata nel file "scoresIndexesCorrelations2019.html".

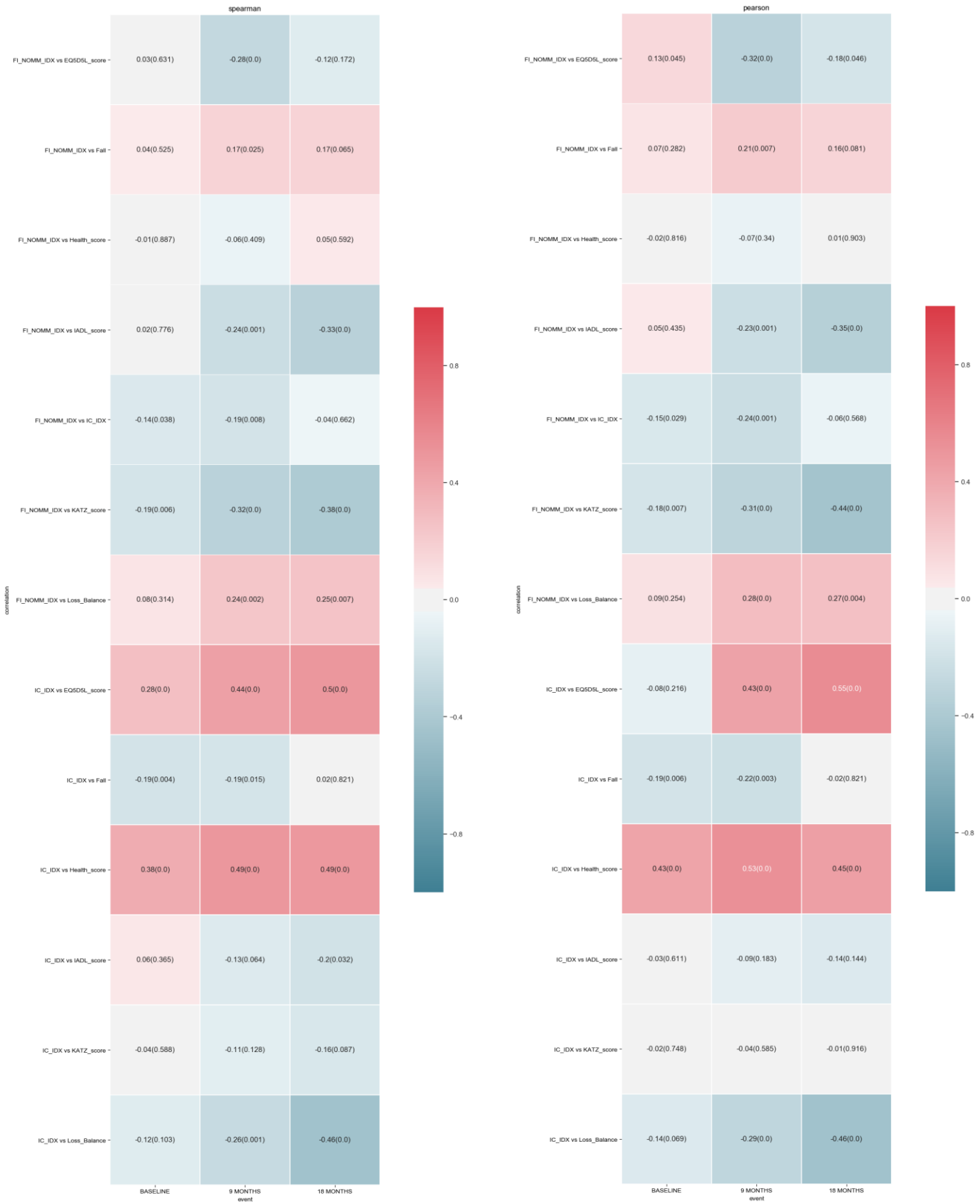


Figura 7-1 Heatmap correlazioni

7.2 Considerazioni sulle correlazioni

Per analizzare i valori delle correlazioni ci basiamo su alcune soglie che definiscono la forza della correlazione diretta.

- Se $0 < \rho_{XY} < 0{,}3$ si ha una correlazione debole.
- Se $0{,}3 < \rho_{XY} < 0{,}7$ si ha una correlazione moderata.
- Se $0{,}7 < \rho_{XY} < 1$ si ha una correlazione forte.

Analogamente valgono le stesse condizioni per la correlazione inversa.

Dal grafico precedente è possibile vedere che l'indice di capacità intrinseca correla direttamente e in modo moderato con EQ5D5L score e Health score.

Per quanto riguarda le altre correlazioni, soprattutto quella tra FI NO MM e IC, per le quali inizialmente ci si aspettava valori alti di correlazione, si sono rivelate deboli.

Conclusioni

Nel corso di questo studio ho dovuto affrontare diverse problematiche, tra le quali imparare a utilizzare al meglio le librerie Python con le quali ho realizzato i diversi script, imparare a individuare quando un dato presenta delle problematiche e in tal caso risolverle, imparare a comprendere le necessità dei medici e fornire loro soluzioni in grado di aiutarli.

Una importante riflessione che si sarebbe dovuta fare prima di iniziare lo studio, ma che è stata tralasciata, creando tutti gli errori risolti in fase di pulizia, doveva essere quella di creare una web app o comunque un sistema per l'inserimento dei dati clinici che vincolasse il medico a utilizzare determinate unità di misura e valori in specifici range, così da poter ridurre al minimo le problematiche riscontrate sui dati.

Nel complesso lo studio si è rivelato utile, i valori di IC correlano con i valori di Health score e EQ5D5L score, ma non presentano correlazioni con i valori di FI, a causa delle diversità nelle condizioni di salute dei pazienti.

Come già detto la mia tesi si limitava alle prime tre fasi della data science, lo studio dopo di me è stato continuato con la creazione di un sistema basato su machine learning in grado di effettuare delle previsioni riguardo i possibili stati di salute futuri dei pazienti basandosi sui precedenti, così da poter prevenire future patologie fisiche o psicologiche.

Lo studio è stato concluso nell'estate 2019 con la pubblicazione dei seguenti abstract:

- Intrinsic capacity but not frailty predicts functional status in PLWH: a multi-centre prospective study.[29]
- Fitness tracking wearable devices and a dedicated smart phone app (MySAwH App) to predict quality of life in PLWH: a multi-centre prospective study.[30]

Bibliografia

[1] "Dato - Wikipedia." [Online]. Available: https://it.wikipedia.org/wiki/Dato.

[2] S. Maneth and A. Poulovassilis, "Data science," *Computer Journal*, vol. 60, no. 3. Oxford University Press, pp. 285–286, 01-Mar-2017.

[3] A. De Mauro, M. Greco, and M. Grimaldi, "A formal definition of Big Data based on its essential features," *Libr. Rev.*, vol. 65, no. 3, pp. 122–135, Apr. 2016.

[4] C. Snijders, U. Matzat, and U. Reips, "'Big Data': Big Gaps of Knowledge in the Field of Internet Science," *Int. J. Internet Sci.*, 2013.

[5] D. Laney, "3D Data Management: Controlling Data Volume, Velocity, and Variety.," *Appl. Deliv. Strateg.*, vol. 949, no. February 2001, p. 4, 2001.

[6] P. Saporito, "The 5 V's of Big Data," *Best's Rev.*, vol. 7, p. 38, 2013.

[7] J. S. Saltz and J. M. Stanton, *An introduction to data science*. .

[8] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Mag.*, vol. 17, no. 3, pp. 37–53, Sep. 1996.

[9] E. D. Reilly, A. Ralston, and D. Hemmendinger, *Encyclopedia of computer science.* Wiley, 2003.

[10] C. H. Yu, "Exploratory Data Analysis," *Oxford Bibliogr.*, p. 4, 2017.

[11] S. Few, "EENIE , MEENIE , MINIE , MOE : Selecting the Right Graph for Your Message," *Intell. Enterp.*, 2004.

[12] "What is NumPy? — NumPy v1.18.dev0 Manual." [Online]. Available: https://www.numpy.org/devdocs/user/whatisnumpy.html.

[13] "Introduction — SciPy v1.3.0 Reference Guide." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/tutorial/general.html.

[14] "scipy.stats.pearsonr — SciPy v1.3.0 Reference Guide." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html

#r8c6348c62346-1.

[15] “scipy.stats.spearmanr — SciPy v1.3.0 Reference Guide.” [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html.

[16] “Package overview — pandas 0.24.2 documentation.” [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html.

[17] “History — Matplotlib 3.1.1 documentation.” [Online]. Available: https://matplotlib.org/users/history.html.

[18] “An introduction to seaborn — seaborn 0.9.0 documentation.” [Online]. Available: https://seaborn.pydata.org/introduction.html.

[19] “Python Graphing Library, Plotly.” [Online]. Available: https://plot.ly/python/.

[20] “Dash User Guide and Documentation - Dash by Plotly.” [Online]. Available: https://dash.plot.ly/introduction.

[21] M. Orsini, M. Pacchioni, A. Malagoli, and G. Guaraldi, “My smart age with HIV: An innovative mobile and IoMT framework for patient’s empowerment,” in *RTSI 2017 - IEEE 3rd International Forum on Research and Technologies for Society and Industry, Conference Proceedings*, 2017.

[22] A. Caselgrandi, A. Malagoli, J. Milic, M. Bloch, E. Spencer, B. Gallagher, G. Lui, C. Cheung, M. Mancini, V. Masi, E. Bardi, M. Corni, M. Menozzi, S. Zona, F. Carli, C. Mussini, and G. Guaraldi, “Improvement of Intrinsic Capacity in Older Adults Living with HIV through a health promotion resource,” in *HIV and Ageing*, 2018.

[23] K. Rockwood and A. Mitnitski, “Frailty in relation to the accumulation of deficits,” *Journals of Gerontology - Series A Biological Sciences and Medical Sciences*, vol. 62, no. 7. Gerontological Society of America, pp. 722–727, 2007.

[24] I. Araujo De Carvalho, C. Martin, M. Cesari, Y. Sumi, J. A. Thiyagarajan, and J. Beard, “Operationalising the concept of intrinsic capacity in clinical settings.”

[25] M. P. Lawton and E. M. Brody, “Assessment of older people: Self-maintaining and instrumental activities of daily living,” *Gerontologist*, 1969.

[26] and M. J. (1963). S. of illness in the aged: T. index of A. A. Katz S, AB Ford, RW Moskowitz, BA Jackson and standardized measure of biological and psychosocial function. 185(12): 914-91, “functional status assessment:The Katz ADL Index,” *J. Am. Med. Assoc.*, vol. 185, no. 12, pp. 914–991, 1963.

[27] M. Herdman, C. Gudex, A. Lloyd, M. Janssen, P. Kind, D. Parkin, G. Bonsel, and X. Badia, “Development and preliminary testing of the new five-level version of EQ-5D (EQ-5D-5L),” *Qual. Life Res.*, vol. 20, no. 10, pp. 1727–1736, Dec. 2011.

[28] M. Bergner, “Measurement of health status,” *Med. Care*, vol. 23, no. 5, pp. 696–704, 1985.

[29] G. Guaraldi, M. Orsini, A. Caselgrandi, A. Malagoli, F. D’Imprima, J. Milic, F. Ghinelli, R. Martoglia, F. Mandreoli, D. Ferrari, G. Liu, and M. Bloch, “Intrinsic capacity but not frailty predicts functional status in PLWH: a multi-centre prospective study,” in *10th International Workshop on HIV & Aging*, 2019.

[30] G. Guaraldi, M. Orsini, A. Caselgrandi, A. Malagoli, F. D’Imprima, J. Milic, F. Ghinelli, R. Martoglia, F. Mandreoli, D. Ferrari, G. Liu, and M. Bloch, “Fitness tracking wearable devices and a dedicated smart phone app (MySAwH App) to predict quality of life in PLWH: a multi-centre prospective study,” in *17th European AIDS Conference (EACS)*, 2019.

Appendice

Script pulizia dati clinici

```
def ClinicalDataCleaner(file_clinical_data):
    """Methods that clean data keeping only necessary parameters."""
    # Define clean data filename
    fcl = file_clinical_data
    fsubject = 'data/PatientBasicData2019.xlsx'
    fcplclean = 'data/Clinic_clean_data2019.xlsx'
    fccounts = 'data/Clinic_Counts2019.xlsx'

    # Read from clinical file
    visit = pd.read_excel(fcl,
                          sheet_name='VISIT - ungrouped',
                          header=1)[['study_subject_id',
                                     'event_name'] + clinical_param]
    registration = pd.read_excel(fcl,
                                 sheet_name='REGISTRATION',
                                 header=1)[['study_subject_id'] +
                                     registration_param]

    # File with pressures values and weight loss
    datap = pd.read_excel(fcl,
                          sheet_name='Press_Weight',
                          index_col=0
                          ).rename(columns={
                                     'IDPaziente': 'id_patient',
                                     'Pa Min': 'PDiast',
                                     'Pa Max': 'PSist',
                                     'CaloPesoInv': 'WeightLoss'})

    # Read drugs file and remove space before Drugs column label
    datad = pd.read_excel(fcl,
                          sheet_name='VISIT - Metabolic Therapy',
                          header=1).rename(columns={
                                     ' Drugs': 'Drugs'})[[
                                             'event_name',
                                             'study_subject_id',
                                             'Drugs']]

    # Get patients basic data, if not present call script that create
    # it
    if os.path.isdir(fsubject):
        subject = pd.read_excel(fsubject,
                                parse_dates=['subject_date_of_birth'])
    else:
        subject = AllPatientsBasicData(fcl, True)

    # Add weight an pressure data
    datap.id_patient = datap.id_patient.astype('str')
```

```
# Cleaning dates
visit[' Date'] = clean_dates(visit[' Date'])
registration['HIV_from'] = clean_dates(registration['HIV_from'])
registration[' ARVT_Start_Date'] =
    clean_dates(registration[' ARVT_Start_Date'])

# Merging data
datar = pd.merge(subject,
                 registration,
                 on='study_subject_id')
datac = pd.merge(datar,
                 visit,
                 on='study_subject_id')
datac = pd.merge(datac,
                 datap,
                 on=['id_patient', 'event_name'],
                 how='left')

# Remove space before column titles
datac.columns = [c.lstrip() for c in datac.columns]

# Drop drugs null
datad = datad[datad.Drugs != '(null)']
# Merging drugs data to subject to get id_patient and site_name
datad = pd.merge(subject[['site_name',
                          'id_patient',
                          'study_subject_id']],
                 datad,
                 on='study_subject_id')

# Drop study_subject_id from tables
datad = datad.drop(columns=['study_subject_id'])
datac = datac.drop(columns=['study_subject_id'])

# Remove value 999 and '(null)'
datac = datac.applymap(
           lambda x: -1 if str(x) in ['999', '(null)'] else x)


# Method is applied on numeric columns
for name in list(datac.columns)
    if name in ClinicalParameterToClean:
        # Clean data
        datac[name] = eval('clean_' + name
           + '(datac[name].astype(float))')

# Remove outliers in Yes/No columns
for name in list(datac.columns):
    if 'label' in name:
        datac[name] = datac[name].map(
           lambda x: True if 'Yes' in str(x)
                          or 'Positive' in str(x)
                          else False if 'No' in str(x)
                          or 'Negative' in str(x) else -1)

# Create a DataFrame for values
data_count = datac.groupby(['event_name',
                            'site_name']).apply(
```

```
        lambda x: x.apply(
            lambda y: pd.Series({'GOOD': y[~y.isin([-1, -2])].count(),
                                 'NOTGOOD': y[y == -2].count(),
                                 'ABSENT': y[y == -1].count()})))

    del data_count['event_name']
    del data_count['site_name']
    data_count = data_count.reset_index().rename(
                      columns={'level_2': 'status'})

    # Replace value used four counts with nan
    datac = datac.replace([-1, -2], np.nan)


    # Write counts on file
    data_count.to_excel(fccounts)


    # Write clean data on file
    writer = pd.ExcelWriter(fcplclean)
    datac.to_excel(writer, sheet_name='Clinical_Data')
    datad.to_excel(writer, sheet_name='Drugs_Data')
    writer.save()
```

Script 8 Pulizia dati clinici

Script pulizia dati psicologici

```
def PsychologicalDataCleaner(file_psychological_data):
    """Method that clean and partially transform psychological
       data."""
    # Define clean data filename
    fpl = file_psychological_data
    fplclean = 'data/Psycho_clean_data2019.xlsx'

    # Open output file
    writer = pd.ExcelWriter(fplclean)

    def clean_and_save(data, writer, sheet):
        """Function that clean dates and save excel."""
        data = data.drop(columns=['id_visit', 'ordinal'])
        data['id_patient'] = data['id_patient'].map(
                lambda x: x.strip())
        data['date'] = clean_dates(data['date'])

        data = numerate_dates(data)

        # Rename Clinical Center in site_name
        data = data.rename(columns={'Clinical Center': 'site_name'})
        # Save data on excel
```

```
        data.to_excel(writer, sheet_name=sheet)

    # Data EMA Physical
    datap_ema_phys = pd.read_excel(fpl, sheet_name='ema physical')
    datap_ema_phys = datap_ema_phys.drop(columns='ema_physical_q02')
    datap_ema_phys['ema_physical_q01'] =
        datap_ema_phys['ema_physical_q01'].map(clean_ema_physical)
    datap_ema_phys =
        datap_ema_phys.rename(columns={
                        'ema_physical_q01': 'ema_physical'})
    clean_and_save(datap_ema_phys, writer, 'ema_physical')
    del datap_ema_phys

    # Data EMA Mood
    datap_ema_mood = pd.read_excel(fpl, sheet_name='ema mood neg')
    datap_ema_mood = datap_ema_mood.drop(columns=['id_visit',
                                                  'ordinal',
                                                  'Clinical Center'])
    datap_ema_mood = pd.merge(
        pd.read_excel(fpl,
                       sheet_name='ema mood pos'),
                        datap_ema_mood,
                        on=['id_patient', 'date'])
    for i in range(1, 11):
        datap_ema_mood['ema_mood' + str(i)] =
                datap_ema_mood['ema_mood_pos_q' + "%.2d" % i].map(
                        clean_ema_mood_question)
    for i in range(1, 11):
        datap_ema_mood['ema_mood' + str(i+10)] =
                datap_ema_mood['ema_mood_neg_q' + "%.2d" % i].map(
                        clean_ema_mood_question)

    datap_ema_mood = datap_ema_mood[['id_patient', 'date'] +
                                    ['ema_mood' + str(i)
                                      for i in range(1, 21)] +
                                    ['Clinical Center',
                                      'id_visit',
                                      'ordinal']]
    clean_and_save(datap_ema_mood, writer, 'ema_mood')
    del datap_ema_mood

    # Data EMA Stress
    datap_ema_stress = pd.read_excel(fpl, sheet_name='ema stress')
    datap_ema_stress = datap_ema_stress.rename(columns={
                                      'answer_date': 'date',
                                      'option_text': 'ema_stress',
                                      'patient_code': 'id_patient'})
    clean_and_save(datap_ema_stress, writer, 'ema_stress')
    del datap_ema_stress

    # Data EMA Social
    datap_ema_social = pd.read_excel(fpl, sheet_name='ema social')
    datap_ema_social['ema_social_q01'] =
        datap_ema_social['ema_social_q01'].map(clean_ema_social_home)
    datap_ema_social['ema_social_q02'] =
        datap_ema_social['ema_social_q02'].map(clean_ema_social_alone)
    datap_ema_social['ema_social_q03'] =
        datap_ema_social['ema_social_q03'].map(
```

```
            clean_ema_social_socialization)
datap_ema_social =
    datap_ema_social.rename(columns={
                    'ema_social_q01': 'ema_social_home',
                    'ema_social_q02': 'ema_social_alone',
                    'ema_social_q03': 'ema_social_socialization'})
clean_and_save(datap_ema_social, writer, 'ema_social')
del datap_ema_social

# Data EMA Eat
datap_ema_eat = pd.read_excel(fpl, sheet_name='ema eating')
for i in [1, 2]:
    datap_ema_eat['ema_eat' + str(i)] =
            datap_ema_eat['ema_eating_q' + "%.2d" % i].map(
                    clean_ema_eating_binge)
for i in range(3, 10):
    datap_ema_eat['ema_eat' + str(i)] =
            datap_ema_eat['ema_eating_q' + "%.2d" % i].map(
                    clean_ema_eating_craving_lossapp)
for i in [10, 11]:
    datap_ema_eat['ema_eat' + str(i)] =
            datap_ema_eat['ema_eating_q' + "%.2d" % i].map(
                    clean_ema_eating_lasttwo)

datap_ema_eat =
    datap_ema_eat[['id_patient', 'date'] + ['ema_eat' + str(i)
            for i in range(1, 12)] + ['Clinical Center',
                                      'id_visit',
                                      'ordinal']]
clean_and_save(datap_ema_eat, writer, 'ema_eat')
del datap_ema_eat

# Data EMA Sleep
datap_ema_sleep = pd.read_excel(fpl, sheet_name='ema sleep')
datap_ema_sleep['ema_sleep_q01'] =
    datap_ema_sleep.apply(clean_sleep_quality, axis='columns')
datap_ema_sleep['ema_sleep_q02'] =
    datap_ema_sleep.apply(clean_sleep_quantity, axis='columns')
datap_ema_sleep = datap_ema_sleep.rename(columns={
                            'ema_sleep_q01': 'ema_sleep_quality',
                            'ema_sleep_q02': 'ema_sleep_quantity'})
clean_and_save(datap_ema_sleep, writer, 'ema_sleep')
del datap_ema_sleep

# Data EMA Smoke
datap_ema_smoke = pd.read_excel(fpl, sheet_name='ema smoke')
datap_ema_smoke['option_text'] =
    datap_ema_smoke.apply(clean_smoke, axis='columns')
datap_ema_smoke = datap_ema_smoke.rename(columns={
                            'option_text': 'ema_smoke',
                            'answer_date': 'date',
                            'patient_code': 'id_patient'})
clean_and_save(datap_ema_smoke, writer, 'ema_smoke')
del datap_ema_smoke

# Data IPA
datap_ipa = pd.read_excel(fpl, sheet_name='IPAq')
datap_ipa = datap_ipa.drop(columns=['ipaq2a', 'ipaq3a', 'ipaq4a'])
datap_ipa[['ipaq1', 'ipaq2', 'ipaq3', 'ipaq4']] =
```

```
        datap_ipa[['ipaq1', 'ipaq2', 'ipaq3', 'ipaq4']].applymap(
        lambda cell: [int(s) for s in cell if s.isdigit()][0])
    clean_and_save(datap_ipa, writer, 'ipa')
    del datap_ipa

    # Data Share
    for i in range(1, 6):
        datas = pd.read_excel(fpl, sheet_name='Share0' + str(i))
        datas['ema_share' + str(i)] =
                eval("datas.apply(clean_share" + str(i) + ",
                      axis='columns')")
        if i in [2, 4]:
            datas['ema_new_share' + str(i)] =
                eval("datas.apply(clean_new_share" + str(i) + ",
                      axis='columns')")
            if i == 2:
                datas['ema_share2A'] =
        datas['how_satisfied_are_you_with_your_life_in_general'].map(
        clean_share2A)
                datas = datas[['id_patient',
                               'date',
                               'ema_share' + str(i),
                               'ema_new_share' + str(i),
                               'ema_share2A',
                               'Clinical Center',
                               'id_visit',
                               'ordinal']]
            elif i == 4:
                datas['i_felt_lonely'] = datas['i_felt_lonely'].map(
                                                clean_i_felt_lonely)
                datas = datas[['id_patient',
                               'date',
                               'ema_share' + str(i),
                               'ema_new_share' + str(i),
                               'Clinical Center',
                               'i_felt_lonely',
                               'id_visit',
                               'ordinal']]
        else:
            if i == 5:
                datas['ema_share5A'] =
                       datas['how_often_do_you_have_conflict_
                       with_friends_coworkers_acquainta'].map(
                               clean_share5A)
                datas = datas[['id_patient',
                               'date',
                               'ema_share' + str(i),
                               'ema_share5A',
                                'Clinical Center',
                               'id_visit', 'ordinal']]
            else:
                datas = datas[['id_patient', 'date',
                               'ema_share' + str(i), 'Clinical Center',
                               'id_visit', 'ordinal']]
        clean_and_save(datas, writer, 'ema_share' + str(i))

    # Data Neuro
    datap_neuro = pd.read_excel(fpl, sheet_name='Neuro')
    datap_neuro['neuro_memory1_q01'] =
```

```
        datap_neuro['neuro_memory1_q01'].map(clean_neuro)
    datap_neuro['neuro_memory2_q02'] =
        datap_neuro['neuro_memory2_q02'].map(clean_neuro)
    datap_neuro['neuro_language'] =
        datap_neuro['neuro_language'].map(clean_neuro)
    datap_neuro = datap_neuro.rename(columns={
                                'neuro_memory1_q01': 'ema_neuro1',
                                'neuro_memory2_q02': 'ema_neuro2',
                                'neuro_language': 'ema_neuro3'})
    clean_and_save(datap_neuro, writer, 'ema_neuro')
    del datap_neuro

    # Close excel file
    writer.save()
```

Script 9 Pulizia dati psicologici

Script pulizia score clinici

```
def ClinicalScoresCleaner(file_clinical_data):
    # Define clean data filename
    fcl = file_clinical_data
    fsubject = 'data/PatientBasicData2019.xlsx'
    fcsclean = 'data/Clinic_clean_scores2019.xlsx'

    # Read from clinical file
    scores = pd.read_excel(fcl,
                           sheet_name='QUESTIONNAIRE - ungrouped',
                           header=1)[['study_subject_id',
                                      'event_name'] + clinical_scores]

    if os.path.isdir(fsubject):
        subject = pd.read_excel(fsubject,
                                parse_dates=['subject_date_of_birth'])
    else:
        subject = AllPatientsBasicData(fcl, True)

    # Remove invalid values
    scores = scores.applymap(
        lambda x: np.nan if '(null)' in str(x)
                         or 'unknown' in str(x) or '999' in str(x)
                         else x)


    # Merging data
    datacs = pd.merge(subject, scores, on='study_subject_id')

    # Remove space before column titles
    datacs.columns = [c.lstrip() for c in datacs.columns]

    # Convert values in Health Score in range 0-10
    datacs['Health_score'] =
        datacs['Health_score'].astype(float).map(
                lambda x: x if 0 <= x <= 10 or x == np.nan
```

```
                                else np.round(x/10, 0))

    # Rename columns with shorter names
    datacs = datacs.rename(columns={
        'In_the_past_6_months_have_you_had_a_fall__label': 'Fall',

        'In_the_past_6_months_have_you_been_concerned_with_losing
        _your_balance_and_falling_while_doing_your_usual_daily_
        activities__label': 'Loss_Balance'})

    # Replace fall data with True/False
    datacs['Fall'] = datacs['Fall'].map(
        lambda x: True if x == 'Yes'
                        else False if x == 'No' else np.nan)
    datacs['Loss_Balance'] = datacs['Loss_Balance'].map(
        lambda x: True if x == 'Yes'
                        else False if x == 'No' else np.nan)
    datacs['EQ5D5L_score'] = datacs['EQ5D5L_score'].astype(float)
    datacs['IADL_score'] = datacs['IADL_score'].astype(float)
    datacs['KATZ_score'] = datacs['KATZ_score'].astype(float)

    # Drop study id
    datacs = datacs.drop(columns=['study_subject_id'])

    # Write data on file
    writer = pd.ExcelWriter(fcsclean)
    datacs.to_excel(writer, sheet_name='clinical_scores')
    writer.save()
```

Script 10 Pulizia score clinici

Script web app dati

```
import dash
import dash_table
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import plotly.graph_objs as go
import numpy as np

# Setting style sheet
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

print('Loading data...')
# File for input
# Clinical data file
Clinical = pd.ExcelFile('data/Clinic_clean_data2019.xlsx')
clinical_data = pd.read_excel(Clinical,
                              sheet_name='Clinical_Data',
                              index_col=0,
                              parse_dates=['subject_date_of_birth'])
```

```
# Psycho data
Psychological = pd.ExcelFile('data/Psycho_clean_data2019.xlsx')

# Employment data
Employment = pd.ExcelFile('data/Employ_clean_data2019.xlsx')

# Garmin data
Garmin = pd.ExcelFile('data/Garmin_clean_data2019.xlsx')

# Create a Series with patients nationalities
Nationality = clinical_data[['id_patient',
        'site_name']].drop_duplicates().set_index('id_patient')
nationality_colors = {'Modena': 'rgb(0, 0, 254)',
                      'Hong Kong': 'rgb(255, 0, 0)',
                      'Sydney': 'rgb(255, 159, 0)'}

# Create a Series with patient birthday year
BirthYear = clinical_data[['id_patient',
                           'subject_date_of_birth']
                         ].drop_duplicates().set_index('id_patient')
BirthYear['birth_year'] =
        BirthYear['subject_date_of_birth'].map(lambda x: x.year)
BirthYear = BirthYear.drop(columns=['subject_date_of_birth'])
# Birth year coloring mode variables
year_range =
        BirthYear.quantile([0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1],
                           axis=0).birth_year.values
birth_colors = {ym: 'rgb(0,' + str(col) + ',0)' for ym, col
                in zip(year_range, np.arange(0,
                                             256,
                                             255/len(year_range)))}

# Create a Series with patient sex
Sex = clinical_data[['id_patient',
                     'subject_sex']
                    ].drop_duplicates().set_index('id_patient')

# All plottable index
data_not_to_plot = ['id_patient', 'subject_date_of_birth',
                    'subject_sex', 'site_name', 'event_name',
                    'date', 'month', 'Clinical Center', 'HIV_from',
                    'ARVT_Start_Date', 'Date',
                    'ema_social_socialization']
data_to_plot = dict()

# Event based data
for df in [clinical_data]:
    for col in df.columns:
        if col in data_not_to_plot:
            continue
        else:
            data = pd.pivot_table(data=df,
                                  index='id_patient',
                                  columns='event_name', values=col)
            data = data[data.columns[::-1]]
            data_to_plot.update({col: data})


# Month based data
```

```
for file in [Psychological, Employment]:
    for sheet in file.sheet_names:
        df = pd.read_excel(file, sheet_name=sheet, index_col=0)
        for col in df.columns:
            if col in data_not_to_plot:
                continue
            else:
                data_to_plot.update(
                        {col: pd.pivot_table(data=df,
                                             index='id_patient',
                                             columns='month',
                                             values=col)})

# Multi month based data
for file in [Garmin]:
    for sheet in file.sheet_names:
        df = pd.read_excel(file, sheet_name=sheet, index_col=0)
        df = df.groupby(['id_patient', 'month']).sum().reset_index()
        for col in df.columns:
            if col in data_not_to_plot:
                continue
            else:
                data_to_plot.update({col: pd.pivot_table(
                                            data=df,
                                             index='id_patient',
                                             columns='month',
                                             values=col)})

# Delete this DFs
del Clinical, Psychological, Employment, Garmin
print('Data fully loaded!')

# App Layout
app.layout = html.Div([
    # First menus row
    html.Div([
        # Data chooser dropdown menu
        html.Div([
            html.P('DATA FILTER'),
            dcc.Dropdown(
                id='crossfilter_select_data',
                options=[{'label': i, 'value': i}
                                for i in list(data_to_plot.keys())],
                value=list(data_to_plot.keys())[0]
            ),
            ],
            style={'width': '49%', 'display': 'inline-block'}),
        # Color filter dropdown menu
        html.Div([
            html.P('COLOR FILTER'),
            dcc.Dropdown(
                id='crossfilter_select_coloring',
                options=[{'label': 'Patient Based',
                          'value': 'PB'},
                         {'label': 'Nationality Based',
                          'value': 'NB'},
                         {'label': 'Age Based', 'value': 'AB'}],
                value='PB'
                ),
```

```
            ],
            style={'width': '49%', 'display': 'inline-block'}),
    ], style={
        'borderBottom': 'thin lightgrey solid',
        'backgroundColor': 'rgb(250, 250, 250)',
        'padding': '10px 5px'
    }),

    # Second menus row
    html.Div([
        # Nationality filter dropdown menu
        html.Div([
            html.P('INCLUDE COUNTRY'),
            dcc.Dropdown(
                id='crossfilter_include_countries',
                options=[{'label': i, 'value': i}
                       for i in Nationality['site_name'].unique()],
                value=Nationality['site_name'].unique(),
                multi=True
                ),
            ],
            style={'width': '49%', 'display': 'inline-block'}),
        # Birth year range filter
        html.Div([
            html.P('BIRTH YEAR RANGE'),
            dcc.RangeSlider(
                id='crossfilter_age_range',
                min=BirthYear.min().values[0],
                max=BirthYear.max().values[0],
                step=1,
                value=[BirthYear.min().values[0],
                       BirthYear.max().values[0]],
                marks={int(y): '{}'.format(int(y))
                       for y in np.arange(BirthYear.min().values[0],
                               BirthYear.max().values[0], 5)},
                ),
            ],
            style={'width': '47%',
                   'margin': '15px',
                   'display': 'inline-block'}),
    ], style={
        'borderBottom': 'thin lightgrey solid',
        'backgroundColor': 'rgb(250, 250, 250)',
        'padding': '10px 5px'
    }),

    # Third menus row
    html.Div([
        # Sex filter dropdown menu
        html.Div([
            html.P('INCLUDE SEX'),
            dcc.Dropdown(
                id='crossfilter_include_sex',
                options=[{'label': i, 'value': i}
                               for i in Sex.subject_sex.unique()],
                value=Sex.subject_sex.unique(),
                multi=True
            ),
        ],
```

```
            style={'width': '49%', 'display': 'inline-block'}),
    ], style={
        'borderBottom': 'thin lightgrey solid',
        'backgroundColor': 'rgb(250, 250, 250)',
        'padding': '10px 5px'
    }),

    # Color legend
    html.Div(id='live_legend', style={
                                'borderBottom': 'thin lightgrey solid',
                                'backgroundColor': 'rgb(250, 250,250)',
                                'padding': '10px 5px'
                                        }),

    # Graph plot
    html.Div([
        dcc.Graph(
            id='crossfilter_data_plot',
        )
    ], style={'width': '100%',
              'display': 'inline-block',
              'padding': '0 20'}),

    # Table plot
    html.Div(id='table_plot', style={
        'backgroundColor': 'rgb(250, 250, 250)',
        'width': '100%',
        'display': 'inline-block',
    }),

    html.Div(style={
        'marginBottom': '100px',
        'width': '100%',
        'display': 'inline-block',
    })
])


@app.callback(
    # This is an app callback, every time you change something in
filters it will reload graph
    [dash.dependencies.Output('crossfilter_data_plot', 'figure'),
     dash.dependencies.Output('live_legend', 'children'),
     dash.dependencies.Output('table_plot', 'children')],
    [dash.dependencies.Input('crossfilter_select_data', 'value'),
     dash.dependencies.Input('crossfilter_select_coloring', 'value'),
     dash.dependencies.Input('crossfilter_include_countries','value'),
     dash.dependencies.Input('crossfilter_age_range', 'value'),
     dash.dependencies.Input('crossfilter_include_sex', 'value'), ])
def update_graph(select_data, select_coloring, include_countries,
                 age_range, include_sex):
    # Function that create figure to plot
    # Get data that user select
    dtp = data_to_plot.get(select_data)

    # Create a figure
    fig = go.Figure()
    legend_fig = html.P(' ')
    table_fig = html.P(' ')
```

```
# Data for table
table_data = pd.DataFrame(columns=dtp.columns.tolist())


# For each patient in selected data, add a line relative to that
# patient
for index, row in zip(dtp.index, dtp.values):

    # If patient is not in a selected clinical center it will
    # continue
    if index not in Nationality.index or \
             not Nationality.loc[index].values[0]
             in include_countries:
        continue

    # If patient year is not in selected range it will continue
    if not index.split('_')[0] in BirthYear.index or \
            not BirthYear.loc[index].values[0]
            in range(age_range[0], age_range[1]):
        continue

    # If patient sex is not in selected sex it will continue
    if not Sex.loc[index].values[0] in include_sex:
        continue

    # Hover text
    text = ['MONTH/EVENT: {}<br />PATIENT: {}<br />VALUE: {}<br
            />'.format(dtp.columns[i],
                       index,
                       row[i])
            for i in range(len(row))]

    # Dictionary for selected color filter
    marker = dict()
    if select_coloring == 'NB':
        marker.update(dict(
           color=nationality_colors.get(
                   Nationality.loc[index].values[0])))
        table_data =
           table_data.append(
                   pd.Series(
                           row.tolist()+Nationality.loc[
                                   index].values.tolist(),
                            name=index,
                           index=dtp.columns.tolist()
                                   +['Nationality']))
    elif select_coloring == 'AB':
        by = BirthYear.loc[index].values[0]
        array = list(birth_colors.keys())
        idx = int(np.argmin([abs(el - by) for el in array]))
        marker.update(dict(color=birth_colors.get(array[idx])))
        table_data =
           table_data.append(pd.Series(
                   row.tolist() +
                           BirthYear.loc[index].values.tolist(),
                   name=index,
                   index=dtp.columns.tolist() + ['Birth year']))
```

```
        else:
            table_data =
               table_data.append(pd.Series(
                       row.tolist(),
                       name=index,
                       index=dtp.columns.tolist()))

        # Plot patient
        fig.add_trace(go.Scatter(
            x=dtp.columns,
            y=row,
            hoverinfo='text',
            text=text,
            name=index,
            mode='lines+markers',
            marker=marker
        ))

    # Set figure layout
    fig['layout'].update(
        margin=dict(l=50,
                    r=0,
                    b=50,
                    t=50, ),
        autosize=False,
        height=700,
        title=select_data,
        titlefont=dict(size=25),
        hovermode='closest',
    )

    # Row whit mean patient value
    mean_row = table_data[dtp.columns.tolist()].mean().map(
                       lambda x: np.round(x, 2))

    # Row whit counts
    count_row = table_data[dtp.columns.tolist()].count()

    # Color legend setup
    if select_coloring == 'NB':
        # Legend for nationality color mode
        legend = list()
        for string, color in nationality_colors.items():
            legend.append(html.Li([html.Div(
                                     style={'height': '10px',
                                            'width': '10px',
                                            'background-color': color,
                                            'display': 'inline-block'},
                                               ),
                                   html.P(string,
                                     style={'display': 'inline-block',
                                                 })],
                                  style={'list-style': 'none'}))
        legend_fig = [
            html.P('COLOR LEGEND:'),
            html.Ul([html.Div(legend)])]

        if not table_data.empty:
```

```
        # Create count valid columns
        count_col = table_data.groupby('Nationality').count()


        # Select groups based on nationality
        table_data =
           np.round(table_data.groupby('Nationality').mean(), 2)
        # Add count columns to table data df
        for col in count_col.columns:
            table_data[str(col) + ' COUNT'] = count_col[col]

elif select_coloring == 'AB':
    # Legend for birth date color mode
    legend = list()
    for i in range(len(year_range)-1):
        color = birth_colors.get(year_range[i])
        start = year_range[i]
        end = year_range[i+1]
        legend.append(html.Li([html.Div(
                                style={'height': '10px',
                                       'width': '10px',
                                       'background-color': color,
                                       'display': 'inline-block'},
                                          ),
                               html.P(str(start) + '-' + str(end),
                                style={'display': 'inline-block',
                                          })],
                              style={'list-style': 'none'}))
    legend_fig = [
        html.P('COLOR LEGEND:'),
        html.Ul([html.Div(legend)])]

    if not table_data.empty:
        # Create count valid columns
        count_col = table_data.groupby('Birth year').count()
        # Select groups based on birth year
        table_data = np.round(table_data.groupby('Birth year'
                                                 ).mean(), 2)
        # Add count columns to table data df
        for col in count_col.columns:
            table_data[str(col) + ' COUNT'] = count_col[col]
else:
    if not table_data.empty:
        # Add count valid columns
        table_data['count'] = table_data.apply(
                                  lambda x: x.count(), axis=1)
        # Table with all patient statistic data
        table_data = np.round(table_data, 2)

    if not table_data.empty:
        # Set id_patient as name for index column and reset index
        table_data.index =
           table_data.index.set_names(['id_patient'])
        table_data = table_data.reset_index()

    # Standard variation column
    table_data['stdv'] = table_data[dtp.columns.tolist()].apply(
            lambda x: np.round(x.mean() + x.std(), 2), axis=1)
```

```
        # Mean values column
        table_data['mean'] = table_data[dtp.columns.tolist()].apply(
                                lambda x: np.round(x.mean(), 2), axis=1)

        # Delta columns for event based data
        if 'BASELINE' in table_data.columns.tolist():
            table_data['delta 0-9'] = np.round(
               table_data['9 MONTHS'] - table_data['BASELINE'], 2)
            table_data['delta 9-18'] = np.round(
               table_data['18 MONTHS'] - table_data['9 MONTHS'], 2)
            table_data['delta 0-18'] = np.round(
               table_data['18 MONTHS'] - table_data['BASELINE'], 2)

        # Delta columns for month based data
        else:
            table_data['delta 0-9'] = np.round(
               table_data[9] - table_data[1], 2)
            table_data['delta 9-18'] = np.round(
               table_data[18] - table_data[9], 2)
            table_data['delta 0-18'] = np.round(
               table_data[18] - table_data[1], 2)

        # Append rows with counts and mean
        mean_row[table_data.columns[0]] = 'MEAN'
        count_row[table_data.columns[0]] = 'COUNT'
        table_data = table_data.append(mean_row, ignore_index=True)
        table_data = table_data.append(count_row, ignore_index=True)

        # Creating table figure
        table_fig = [
            dash_table.DataTable(
                id='table',
                columns=[{"name": str(i), "id": str(i)}
                       for i in table_data.columns],
                data=table_data.to_dict('records'),
            )
        ]

    return fig, legend_fig, table_fig

# MAIN


if __name__ == '__main__':
    app.run_server(debug=True,
                   port=8051,
                   threaded=True,
                   )
```

Script 11 Web app dati

Script conteggio dati clinici

```
def get_count_clinic_data():
    """Script that create a file with histograms relative to clinical
       data in different periods."""
    cdata = pd.read_excel('data/Clinic_Counts2019.xlsx', index_col=0)
    file_pdf_output = 'plots/Clinic_Counts2019.png'

    # Set figure size and figure parameters
    fig = plt.figure(figsize=(10, 2*(len(cdata.columns)-3) *
                        cdata.event_name.nunique()))
    plt.subplots_adjust(hspace=0.05)

    w = 1
    h = int(cdata.event_name.nunique() / w) + 1
    c = 0
    # Set style
    sns.set(style='darkgrid')

    # For each event create a subset
    for e in np.flip(cdata.event_name.unique()):
        subset = cdata[cdata.event_name == e]

        c += 1
        ax = fig.add_subplot(h, w, c)

        # Group count by site name and create columns for status of
        # values
        ord_count = subset.groupby('site_name').apply(
                lambda x: x.pivot_table(columns=subset[['status']]), )
        ord_count = ord_count.reindex(['GOOD', 'NOTGOOD', 'ABSENT'],
                                      axis=1)
        ord_count = ord_count.reset_index().sort_values('level_1'
                        ).reset_index(drop=True)
        ord_count = ord_count.set_index(['site_name', 'level_1'])

        # Plot values on histogram
        ord_count.plot(kind='barh',
                        stacked=True,
                        width=1,
                        fontsize=14,
                        color=['g', 'r', 'b'],
                        ax=ax)

        # Set xticks, label, title
        ax.set(xticks=(np.arange(0, 170, 10)))
        ax.set_xlabel('COUNT', fontweight='bold', size=20)
        ax.set_ylabel('VALUES PER CLINICAL CENTER',
                        fontweight='bold', size=20)
        ax.set_title(e + ' VALUE COUNTS', fontweight='bold', size=30)
        ax.legend(fontsize=14)

        # Add annotations
        for i, v in enumerate(ord_count.values):
            for j in range(len(v)):
```

```
                if v[j]:
                    ax.text(x=sum(v[:j]) + v[j]/2 - len(str(v[j])),
                            y=i-0.25,
                            s=str(v[j]),
                            color='black',
                            fontweight='bold')
    # Save plot on file
    plt.savefig(file_pdf_output, bbox_inches='tight')
```

Script 12 Conteggio dati clinici

Script conteggio dati psicologici

```
def get_count_psycho_data():
    """Script that create a file with an histogram relative to patient
        count in each month."""
    # Read excel files
    fpd = pd.ExcelFile('data/Psycho_clean_data2019.xlsx')
    fed = pd.ExcelFile('data/Employ_clean_data2019.xlsx')
    file_pdf_output = 'plots/Psycho_Counts2019.png'

    # Create a list with percent size of each histogram
    row_width = list()
    for sheet in fpd.sheet_names:
        nmonth = pd.read_excel(fpd, sheet_name=sheet).month.nunique()
        row_width.append(nmonth)
    row_width.append(pd.read_excel(fed).month.nunique())
    totmonth = sum(row_width)
    row_width = [row/totmonth for row in row_width]

    # Set figure size and figure parameters
    print(row_width)
    fig, ax = plt.subplots(len(fpd.sheet_names) + 1, 1,
                           gridspec_kw={'height_ratios': row_width,
                                        'hspace': 0.02, 'top': 0.80},
                           figsize=(10, len(fpd.sheet_names)*30+30))
    # Clean default figures that plt subplots create
    plt.clf()

    # Table width and height
    w = 1
    h = int(len(fpd.sheet_names) / w) + 1

    # Set style
    sns.set(style='darkgrid')

    c = 0
    # For each sheet create a subplot and plot an histogram
    for sheet in fpd.sheet_names:
        ax[c] = fig.add_subplot(h, w, c+1)
        prepare_and_plot_data(fpd, sheet, ax[c])
        c += 1

    # Plot data relative to employment status in a subplot
```

```
    ax[c] = fig.add_subplot(h, w, c+1)
    prepare_and_plot_data(fed, 'employment_status', ax[c])
    c += 1

    # Save plot on file
    plt.savefig(file_pdf_output, bbox_inches='tight')
```

Script 13 Conteggio dati psicologici

Classe per il calcolo del FI

```
import pandas as pd
import numpy as np
from parameterFINOMMIndex import *


class FINOMMCalculator:
    """Class used to calculate FI NOMM index."""

    def __init__(self):
        # File for input and output
        fcd = pd.ExcelFile('data/Clinic_clean_data2019.xlsx')
        fpd = pd.ExcelFile('data/Psycho_clean_data2019.xlsx')
        self.__file_excel_output = 'data/FI_NOMM_index2019.xlsx'

        # Load excel input files
        self.__cdata = pd.read_excel(fcd,
                                     sheet_name='Clinical_Data',
                                     parse_dates=['HIV_from',
                                      'ARVT_Start_Date', 'Date'])
        for sheet, parameters in fi_NONMM_psycho.items():
            exec('self.data_' + sheet
                 + ' = pd.read_excel(fpd, sheet_name = sheet)'
                 + '[["id_patient", "month"] + parameters]')

        self.__ddata = pd.read_excel(fcd, sheet_name='Drugs_Data')
        self.__fi_NOMM = None

    def get_FINOMM(self):
        """Method that return FI NOMM."""

        def fi_calc(subject_row):
            """Function that calculate a single patient-event FI NOMM
               value"""
            used_var = 0
            val = 0
            xg = subject_row['subject_sex']
            # Parameter calculation
            for k in fi_NOMM_param:
                xki = subject_row[k.strip()]
                if pd.notna(xki):
                    if k in gender_param:
                        val += eval('score_' + k.strip() + '(xki,xg)')
                    else:
```

```
                    val += eval('score_' + k.strip() + '(xki)')
                used_var += 1

        id = subject_row.id_patient
        event = subject_row.event_name
        months = [int(s) for s in event.split() if s.isdigit()][0]
                  if not event == 'BASELINE' else 0

        # Parameter relative to drugs consumption
        if polypharmacyThreshold < self.__ddata[
            (self.__ddata.event_name == event) &
            (self.__ddata.id_patient == id)][
                    'Drugs'].value_counts().sum():
            val += 1
        used_var += 1

        # Parameter relative to psycho data
        for sheet, parameters in fi_NONMM_psycho.items():
            for p in parameters:
                param = eval('self.data_' + sheet
                             + '[ (self.data_' + sheet
                             + '.id_patient == id) &'
                             + '(self.data_' + sheet
                             + '.month == months)]')
                if not param.empty and
                   not np.isnan(param[p].values[0]):
                    if sheet == 'ema_mood':
                        val += score_ema_mood_neg(
                                   param[p].values[0])
                    else:
                        val += param[p].values[0]
                    used_var += 1

        # If number of valid values is bigger than threshold
        # return fi
        if used_var >= fi_NOMM_min_values:
            # Return FI value for row
            return val / used_var
        else:
            print(id)
            return np.nan

    # Add a column with FI NOMM value for each row
    self.__cdata['fi_val'] = self.__cdata.apply(fi_calc, axis=1)

    # Return a pivot table with FI NOMM value for each patient in
    # each period
    self.__fi_NOMM = self.__cdata.pivot_table(index='id_patient',
                                              columns='event_name',
                                              values='fi_val'
                                               ).drop_duplicates()
    # Set column order
    self.__fi_NOMM = self.__fi_NOMM[['BASELINE',
                                     '9 MONTHS',
                                     '18 MONTHS']]

    return self.__fi_NOMM

def save_FINOMM(self):
```

```
        """Method that save FI NOMM on file"""
        if self.__fi_NOMM is not None:
            self.__fi_NOMM.to_excel(self.__file_excel_output)
```

Script 14 Classe per il calcolo del FI

Classe per il calcolo del ICI

```
import pandas as pd
import numpy as np
from parameterICIndex import *


class ICCalculator:
    """Class used to calculate IC index."""

    def __init__(self):
        # File for input and output
        fpbd = pd.ExcelFile('data/PatientBasicData2019.xlsx')
        fed = pd.ExcelFile('data/Employ_clean_data2019.xlsx')
        fpd = pd.ExcelFile('data/Psycho_clean_data2019.xlsx')
        fgd = pd.ExcelFile('data/Garmin_clean_data2019.xlsx')

        self.__file_excel_output = 'data/IC_index2019.xlsx'
        self.__pbdata = pd.read_excel(fpbd)
        self.__edata = pd.read_excel(fed,
                                     sheet_name='employment_status',
                                     parse_dates=['date'],
                                     index_col=0)
        self.__gdata = pd.read_excel(fgd, sheet_name='garmin')
        self.__IC = pd.DataFrame(columns=np.arange(1, NOM+1).tolist())

        for sheet, parameters in ic_parameters.items():
            exec('self.data_' + sheet
                 + ' = pd.read_excel(fpd, sheet_name = sheet)['
                 + '["id_patient", "month"] + parameters]')

    def get_IC(self):
        """Method that return IC."""

        def ic_calc(patient, month):
            """Function that calculate a single patient-event IC
               value"""
            used_var = 0
            val = 0

            # Parameter relative to unemployment
            # Search the first valid going back in months
            j = self.__edata[(self.__edata.id_patient == patient)]
            unemp = np.nan
            c = -1
            while np.isnan(unemp) and month - c > 0:
                c += 1
```

```
            try:
                unemp = j[self.__edata.month == (month - c)][
                    'unemployed'].values[0]
            except IndexError:
                continue
        if not np.isnan(unemp):
            val += unemp
            used_var += 1

        # Parameter relative to ema mood
        for sheet, parameters in ic_parameters.items():

            for p in parameters:
                param = eval('self.data_' + sheet
                             + '[(self.data_' + sheet
                             + '.id_patient == patient) &'
                             + '(self.data_' + sheet
                             + '.month == month)]')
                if not param.empty and
                   not np.isnan(param[p].values[0]):
                    used_var += 1

                    if p in ['ema_mood' + str(i)
                             for i in range(1, 11)]:
                        val += score_ema_mood_pos(
                                   param[p].values[0])
                    elif p in ['ema_mood' + str(i)
                             for i in range(11, 21)]:
                        val += score_ema_mood_neg(
                                   param[p].values[0])
                    elif p in ['ema_eat' + str(i)
                                   for i in range(1, 3)]:
                        val += score_ema_eat_binge(
                                   param[p].values[0])
                    elif p in ['ema_eat' + str(i)
                                   for i in range(3, 11)]:
                        val += score_ema_eat(
                                   param[p].values[0])
                    else:
                        val += eval('score_' + p
                                    + '(param[p].values[0])')

        # Parameters relative to garmin score
        gd = self.__gdata[(self.__gdata.id_patient == patient) &
                          (self.__gdata.month == month)]
        if not gd.empty:
            gd = gd.mean()
            for p in ic_garmin:
                if not np.isnan(gd[p]):
                    val += eval('score_garmin_' + p + '(gd.'
                                + p + ')')
                    used_var += 1

        # If number of valid values is bigger than threshold
        # return fi
        if used_var >= ic_min_values:
            # Return FI value for row
            return 1 - val / used_var
        else:
```

```
                return np.nan

        # Add a column with IC value for each row
        for patient in self.__pbdata.id_patient.values:
            pseries = pd.Series(index=np.arange(1, NOM+1).tolist())
            pseries = pseries.rename(patient)
            for month in range(1, NOM+1):
                pseries[month] = ic_calc(patient, month)

            self.__IC = self.__IC.append(pseries)

        self.__IC.index.name = 'id_patient'

        return self.__IC

    def save_IC(self):
        """Method that save IC on file"""
        if self.__IC is not None:
            self.__IC.to_excel(self.__file_excel_output)
```

Script 15 Classe per il calcolo del ICI

Script per la heamap degli indici

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from matplotlib import colorbar, gridspec
from matplotlib.colors import LinearSegmentedColormap as LSC
from matplotlib.colors import Normalize


# ###SETTINGS###
countries = ['Modena', 'Hong Kong', 'Sydney']
indexes = ['FI_NOMM_IDX', 'IC_IDX']
columns = np.arange(1, 18).tolist()

# Normalize negative colorbar values
norm = Normalize(-1, 1)

# Heatmaps colormaps
nationality_colormap = {
        'Modena': [[norm(-1.0), 'black'], [norm(-0.1), 'blue'],
                   [norm(0.1), 'black'], [norm(1.0), 'yellow']],
        'Hong Kong': [[norm(-1.0), 'black'], [norm(-0.1), 'red'],
                      [norm(0.1), 'black'], [norm(1.0), 'cyan']],
        'Sydney': [[norm(-1.0), 'black'], [norm(-0.1), 'orange'],
                   [norm(0.1), 'black'], [norm(1.0), 'purple']]
                        }

# Colorbars colormaps
colorbar_colormap = {'Modena': {'FI_NOMM_IDX': [[0, 'blue'],
```

```
                                                    [1, 'black']],
                                 'IC_IDX': [[0, 'black'],
                                            [1, 'yellow']]},
                     'Hong Kong': {'FI_NOMM_IDX': [[0, 'red'],
                                                  [1, 'black']],
                                   'IC_IDX': [[0, 'black'],
                                              [1, 'cyan']]},
                     'Sydney': {'FI_NOMM_IDX': [[0, 'orange'],
                                                [1, 'black']],
                                 'IC_IDX': [[0, 'black'],
                                            [1, 'purple']]}
                     }


# ###MAIN###
# Load files
FI_NOMM_IDX = pd.read_excel('data/FI_NOMM_index2019.xlsx',
                            index_col=0)
IC_IDX = pd.read_excel('data/IC_index2019.xlsx', index_col=0)

# Patient Basic data
IDX = pd.read_excel('data/PatientBasicData2019.xlsx',
parse_dates=['subject_date_of_birth'])
IDX = IDX[['id_patient', 'site_name']].sort_values(by='id_patient')

# Patient nationality
Nationality = IDX[['id_patient',
'site_name']].drop_duplicates().set_index('id_patient')

# Because FI values are grouped in 9 months groups, expand it
for i in range(1, 27):  # mancano dei mesi
    if i < 9:
        FI_NOMM_IDX[i] = -FI_NOMM_IDX['BASELINE']
    elif i in range(9, 18):
        FI_NOMM_IDX[i] = -FI_NOMM_IDX['9 MONTHS']
    else:
        FI_NOMM_IDX[i] = -FI_NOMM_IDX['18 MONTHS']

FI_NOMM_IDX = FI_NOMM_IDX.drop(columns=['18 MONTHS',
                                        '9 MONTHS',
                                        'BASELINE'])


# Countries data
for country in countries:
    exec('data' + country.replace(' ', '_')
         + ' = pd.DataFrame(columns=columns)')

# For each patient add his indexes rows
for patient, nationality in zip(IDX.id_patient, IDX.site_name):
    if nationality not in countries:
        continue
    for idx in indexes:
        dtp = eval(idx)[columns]
        row = dtp[dtp.index == patient]
        row.index = row.index + '_' + idx

        # If row empty add nan row to DF
        if row.empty:
            row = pd.DataFrame(index=[patient +'_' + idx],
```

```
                                columns=columns)

        # Append row to DF
        data = eval('data' + nationality.replace(' ',
                                                 '_')).append(row)
        exec('data' + nationality.replace(' ', '_') + ' = data')

# Get total number o rows
tot_rows = sum([eval(idx).shape[0] for idx in indexes])

# Row thickness
row_width = []
for country in countries:
    row_width.append(eval('data' + country.replace(' ', '_')).shape[0]
        / tot_rows)

legend_height = 0.01
row_width = [x - legend_height/len(countries) for x in row_width]
row_width = [legend_height] + row_width

# Set seaborn style
sns.set(style='darkgrid')

# Create a linear colormap
cmap = {'Modena': LSC.from_list("",
                nationality_colormap.get('Modena')),
        'Hong Kong': LSC.from_list("",
                nationality_colormap.get('Hong Kong')),
        'Sydney': LSC.from_list("",
                nationality_colormap.get('Sydney')),
        }

# Create a linear colorbar
cb_cmap = {
        'Modena': {'FI_NOMM_IDX': LSC.from_list("",
                colorbar_colormap.get('Modena').get('FI_NOMM_IDX')),
                   'IC_IDX': LSC.from_list("",
                colorbar_colormap.get('Modena').get('IC_IDX'))},
        'Hong Kong': {'FI_NOMM_IDX': LSC.from_list("",
                colorbar_colormap.get('Hong Kong').get('FI_NOMM_IDX')),
                       'IC_IDX': LSC.from_list("",
                colorbar_colormap.get('Hong Kong').get('IC_IDX'))},
        'Sydney': {'FI_NOMM_IDX': LSC.from_list("",
                colorbar_colormap.get('Sydney').get('FI_NOMM_IDX')),
                    'IC_IDX': LSC.from_list("",
                colorbar_colormap.get('Sydney').get('IC_IDX'))},
           }


# Create figure
f, ax = plt.subplots(len(countries)+1, 1,
            gridspec_kw={'height_ratios': row_width,
                         'hspace': 0.04,
                         'top': 0.98},
            figsize=(20, tot_rows))

# Row index
row_idx = 0
```

```
# Add legend subplot
legend = gridspec.GridSpecFromSubplotSpec(len(countries)*2, 1,
                                          subplot_spec=ax[row_idx],
                                          wspace=0.1,
                                          hspace=0.0,
                                          )

# For each country for each index create a colormap
sub_idx = 0
for country in countries:
    for index in indexes:
        axl = plt.Subplot(f, legend[sub_idx])
        axl.get_yaxis().set_ticks([])
        g0 = colorbar.ColorbarBase(
                                orientation='horizontal',
                                cmap=(cb_cmap.get(country).get(index)),
                                ax=axl)
        g0.ax.set_xticklabels([])


        g0.ax.set_ylabel(country+' '+index,
                         rotation=0,
                         horizontalalignment='right')
        f.add_subplot(axl)
        sub_idx += 1

ax[0].yaxis.set_ticks([])
ax[0].set_title('FI NOMM and IC INDEXES',
                fontdict={'fontsize': 24})
row_idx+=1

# For each country DF create an heatmap
for country in countries:
    exec('g' + str(row_idx+1) + ' = sns.heatmap(data'
         + country.replace(' ', '_') + ','
         + 'cmap=cmap.get(country), cbar=False, '
         + 'ax=ax[row_idx], annot=np.abs(data'
         + country.replace(' ', '_') + '.values),)')
    exec('g' + str(row_idx+1) + '.set_ylabel(country)')
    exec('g' + str(row_idx + 1) + '.set_title(country,'
         + 'fontdict={"fontsize":20})')
    row_idx += 1

# Add ticks and labels
for ax in [eval('g' + str(i)) for i in range(2, len(countries)+2)]:
    ax.xaxis.tick_top()
    ax.set_xlabel('MONTHS')
    ax.set_ylabel('ID PATIENT')
    ax.xaxis.set_label_position('top')

# Save on file
plt.savefig('plots/IndexesHeatmap.png', bbox_inches='tight')
```

Script 16 Heatmap degli indici

Classe per le correlazioni

```
import pandas as pd
import numpy as np
import plotly as py
import plotly.graph_objs as go
from plotly import tools
from scipy.stats import pearsonr, spearmanr
import matplotlib.pyplot as plt
import seaborn as sns

methods = ['spearman', 'pearson']
plotout = 'plots/scoresIndexesCorrelations2019.html'
imageout = 'plots/scoresIndexesCorrelations2019.png'


class CorrelateS2I:
    """Class that evaluate correlations between indexes and scores."""

    def __init__(self):
        # Load files
        self.FI_NOMM_IDX = pd.read_excel(
                        'data/FI_NOMM_index2019.xlsx',
                        index_col=0)
        self.IC_IDX = pd.read_excel('data/IC_index2019.xlsx',
                                    index_col=0)
        self.SCORES = pd.read_excel(
                        'data/Clinic_clean_scores2019.xlsx',
                        index_col=0)

        # Open output file
        self.fileout = pd.ExcelWriter(
                        'data/scoresIndexesCorrelations2019.xlsx')

        # Create DF for correlations
        self.correlated_data = pd.DataFrame(
                        columns=['correlation', 'event', 'value',
                                 'method', 'p_value'])

        # Prepare monthly indexes to correlation
        self.IC_IDX['BASELINE'] = self.IC_IDX[1]
        self.IC_IDX['9 MONTHS'] = self.IC_IDX[
                       np.arange(2, 10).tolist()].mean(axis='columns')
        self.IC_IDX['18 MONTHS'] = self.IC_IDX[
                       np.arange(10, 19).tolist()].mean(axis='columns')
        self.IC_IDX = self.IC_IDX[['BASELINE', '9 MONTHS',
                                   '18 MONTHS']]

        # List of needed correlations
        self.data = []
        for method in methods:
          for index in ['FI_NOMM_IDX', 'IC_IDX']:
            for score in ['Fall', 'Loss_Balance',
                          'EQ5D5L_score', 'Health_score',
                          'KATZ_score', 'IADL_score']:
              for event in eval('self.' + index +
```

```
                                '.columns.tolist()'):
                data_to_corr = pd.concat([eval('self.' + index
                                            + '[event]'),


                                            pd.pivot_table(
               data=self.SCORES[self.SCORES.event_name == event],
               index='id_patient',
               values=score)],
               axis=1).dropna(axis=0)
                 x, r = eval(method + 'r(data_to_corr.iloc[:,'
                            + '0].tolist(), data_to_corr.iloc[:,'
                            + '1].tolist())')

                self.correlated_data = self.correlated_data.append(
                      {'correlation': index + ' vs ' + score,
                       'event': event,
                       'value': x,
                       'method': method,
                       'p_value': r},
                      ignore_index=True)

            # Add FI NO MM to IC correlation
            for event in eval('self.' + index + '.columns.tolist()'):
                data_to_corr = pd.concat([self.IC_IDX[event],
                                          self.FI_NOMM_IDX[event]],
                                         axis=1).dropna(axis=0)
                # Correlate indexes
                x, r = eval(method + 'r(data_to_corr.iloc[:,'
                            + '0].tolist(), data_to_corr.iloc[:,'
                            + '1].tolist())')

                # Add correlations results to DF
                self.correlated_data = self.correlated_data.append(
                      {'correlation': 'FI_NOMM_IDX vs IC_IDX',
                       'event': event,
                       'value': x,
                       'method': method,
                       'p_value': r},
                      ignore_index=True)

        # Save on file
        self.correlated_data.to_excel(self.fileout)
        self.fileout.save()

    def plot_seaborn(self):
        """Method that create a correlations static image."""
        # Set style
        sns.set(style="white")

        # Create figure
        f, ax = plt.subplots(len(methods), 1,
                             gridspec_kw={'hspace': 0.03,
                                          'top': 0.80},
                             figsize=(20,
                               self.correlated_data.shape[0]))

        # For each method, create two pivot-tables with events as
```

```
        # columns an correlation as rows
        for r, method in enumerate(
                self.correlated_data.method.unique()):
            # In the first pivot table put correlation values
            data_to_plot = self.correlated_data[
                        self.correlated_data.method == method
                                ].pivot_table(index='correlation',
                                              columns='event',
                                              values='value')[
                                                      ['BASELINE',
                                                       '9 MONTHS',
                                                       '18 MONTHS']]

            # In the second pivot tables put p-value values
            pvalues = self.correlated_data[
                        self.correlated_data.method == method
                                ].pivot_table(index='correlation',
                                             columns='event',
                                             values='p_value')[
                                                      ['BASELINE',
                                                       '9 MONTHS',
                                                       '18 MONTHS']]
            # Round p-value values at third digit
            pvalues = np.around(pvalues.values, 3).astype(str)

            # Create annotations table
            annotations = np.around(data_to_plot.values,
                                    2).astype(str)
            for row in range(len(annotations)):
                for column in range(len(annotations[row])):
                    annotations[row][column] =
                                      annotations[row][column] + '('
                                      + pvalues[row][column] + ')'

            # Set colormap
            cmap = sns.diverging_palette(220, 10, as_cmap=True)

            # Draw the heatmap with the mask and correct aspect ratio
            sns.heatmap(data_to_plot,
                        cmap=cmap,
                        vmin=-1,
                        vmax=1,
                        center=0,
                        square=True,
                        linewidths=.5,
                        cbar_kws={"shrink": .5},
                        xticklabels=True,
                        yticklabels=True,
                        annot=annotations,
                        fmt='',
                        ax=ax[r])
            ax[r].set_title(method)

        # Save plot on file
        plt.savefig(imageout, bbox_inches='tight')

    def plot_plotly(self):
        """Method to create a plotly interactive plot."""
```

```
        # Create a figure
        fig = tools.make_subplots(
               rows=self.correlated_data.method.nunique(), cols=1)

        # Set layout
        fig['layout'].update(
            margin=dict(l=250,
                        r=50,
                        b=0,
                        t=200, ),
            autosize=False,
            height=30*self.correlated_data.shape[0],
            width=150*self.correlated_data.shape[1],
            title='Correlations',
            titlefont=dict(size=30),
            hovermode='closest',
            )

        # Set colormap with white in the middle value
        center = self.correlated_data.value[~np.isnan(
                       self.correlated_data.value)]
        center = 0 - center.min() / (center.max() - center.min())
        colorscale = [[0.0, 'rgb(0,0,220)'],
                      [center, 'rgb(255,255,255)'],
                       [1.0, 'rgb(220,0,0)']]

        # For each correlation methods create a pivot-table
        for r, method in enumerate(
               self.correlated_data.method.unique()):
                       data_to_plot = self.correlated_data[
                       self.correlated_data.method == method
                               ].pivot_table(index='correlation',
                                             columns='event',
                                             values='value')

            # Select value for axis
            X = data_to_plot.index
            Y = data_to_plot.columns
            Z = data_to_plot.values

            # Set hover text for each cell
            text = np.transpose([['CORRELATION: {}<br />MONTH: {}<br
                                  />VALUE: {}'.format(X[i],
                                                      Y[j],
                                                      Z[i][j])
                                 for i in range(len(X))]
                                     for j in range(len(Y))])

            # Set up data for heatmap
            fig.add_trace(go.Heatmap(z=Z.tolist(),
                                     y=X,
                                     x=Y,
                                     colorscale=colorscale,
                                     colorbar=dict(
                                         tickvals=np.around(
                                              np.arange(-1.0, 1.05,
                                                     0.1), decimals=1),

                                      ticktext=np.around(
```

```
                                                np.arange(-1.0, 1.05,
                                                      0.1), decimals=1)),
                                        hoverinfo='text',
                                        text=text,
                                        zmin=-1.0,
                                        zmax=1.0
                                        ),

                        r+1, 1)
            fig['layout']['yaxis' + str(r + 1)].update(
                                         title=method + 'correlations')
            fig['layout']['xaxis' + str(r + 1)]['autorange'] =
                                         'reversed'

        # Plot on a html file
        py.offline.plot(fig, filename=plotout)
```

Script 17 Classe per le correlazioni

Ringraziamenti

Ringrazio tutti coloro che hanno partecipato alla realizzazione di questo studio, in particolare i docenti Riccardo Martoglia e Federica Mandreoli, che mi hanno supportato nel corso del tirocinio e della scrittura della tesi.

Vorrei inoltre ringraziare Francesca D'Imprima, Jovana Milić, Agnese Caselgrandi e Davide Ferrari con i quali ho collaborato per l'estrazione e la pulizia dei dati.