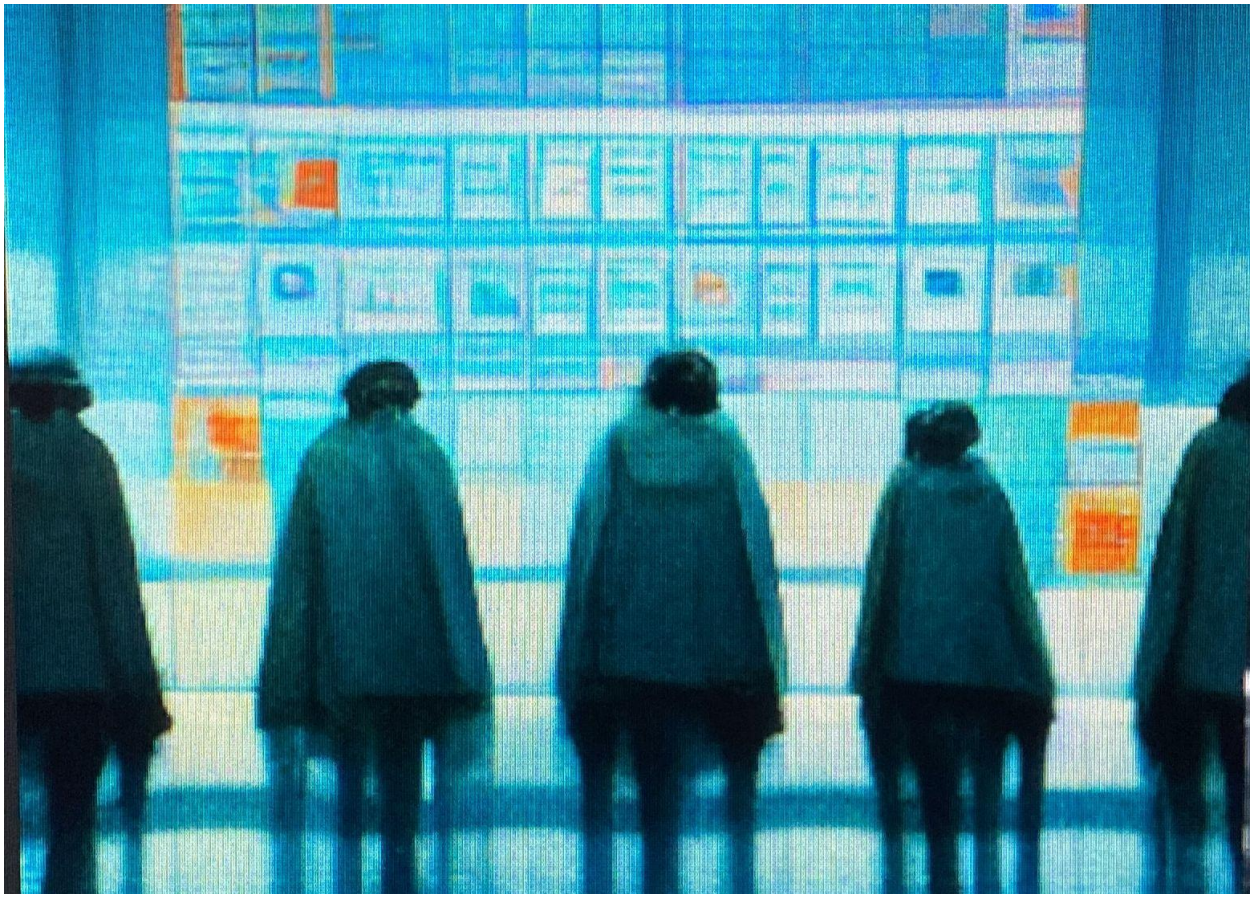
Progetto Covid-19
ANALISTI ANONIMI



Introduzione

Questo logo rappresenta gli "***Analisti Anonimi***". E' stato generato grazie ad un' Intelligenza Artificiale chiamata Midjourney che permette di generare immagini dando in input delle caratteristiche. L'immagine rappresenta ***5 giovani analisti*** pronti a prendere il ***Covid-19*** e a sotterrarlo sotto una ***montagna di dati***.

Descrizione progetto

In questo progetto del corso di **Fondamenti di Informatica** abbiamo approfondito l'utilizzo del linguaggio di programmazione **Python** utilizzandolo per estrarre dei dati riguardanti **l'andamento di varie osservabili** dai bollettini **Covid** quotidiani da un dataset fornito e aggiornato dal 24/02/2020 alla giornata attuale.

L'obiettivo è quello di mettere a disposizione delle **funzioni** che restituiscano: la **media mobile**, la **somma cumulata**, il **valore minimo** ed il **valore massimo** di una qualunque osservabile presente nel **dataset**.

Descrizione delle modalità di lavoro

L'organizzazione del progetto è partita da un ***brainstorming*** iniziale di tutti i componenti degli "***Analisti Anonimi***", andando a definire come sarebbe stato svolto il progetto.

Ogni componente del gruppo ha scelto la funzione che avrebbe dovuto creare, rispettivamente: ***Emanuele Corradi*** ha scelto la ***media mobile***, ***Valerio Valentini*** il ***grafico*** di un'osservabile, ***Daniele Mariani*** la ***somma cumulata*** di un'osservabile, ***Aurelio Caccese*** il ***valore di massimo*** di un'osservabile e ***Stefano De Saraca*** il ***valore di minimo*** di un'osservabile. Tutte queste funzioni analizzano i dati all'interno di un periodo assegnato dall'utente, che può sceglierlo tramite il Menù.

Abbiamo successivamente creato un ***server Discord***, dove abbiamo avuto modo di coordinarci tramite ***riunioni vocali*** e ***presentazioni video*** per la realizzazione del progetto. Inoltre, per rimanere in costante contatto tra di noi, abbiamo creato anche un ***gruppo Whatsapp***, dove ci siamo tenuti quotidianamente aggiornati sull'andamento del progetto.

Ognuno di noi, per prima cosa, ha portato a termine la creazione della propria funzione. Successivamente le abbiamo unite in unico file, usando ***PyCharm*** prestando attenzione a fattori come: l'omologazione delle variabili, l'efficienza dei processi ed il testing del corretto funzionamento. Tutto ciò coordinandoci nella scrittura del codice e collegandoci quotidianamente sul server ***Discord***.

Per evitare errori, ridondanze e inconsistenze di dati, abbiamo creato una ***parte comune***, dove abbiamo importato le librerie necessarie, stabilito i processi e le variabili comuni che ausilieranno le funzioni principali, il menù e il resto del programma.

Per rendere l'esperienza di utilizzo più intuitiva e semplice possibile abbiamo implementato nel programma un ***Menù*** in cui decidere quale funzione utilizzare.

Un punto di forza di quest'ultimo è il possibile ritorno al menù principale in caso l'utente voglia eseguire un'altra analisi.

Per la creazione del ***PowerPoint*** ed il file ***Word***, abbiamo lavorato in sincronia coordinandoci tramite il server Discord, proprio come per la costruzione del programma. Per entrambi abbiamo utilizzato servizi forniti da Google quali ***Google Slides*** e ***Google Docs***. Questi ultimi ci hanno permesso di lavorare simultaneamente al progetto, in modo da completare il proprio lavoro singolarmente per poi cooperare successivamente nella parte in comune.

Una volta terminato il progetto, abbiamo revisionato e ottimizzato il tutto in modo che risultasse pulito, professionale e allo stesso tempo interessante sia nell' esposizione che nella visualizzazione.

Librerie Utilizzate

Per supportare i calcoli che abbiamo definito nel codice del programma abbiamo deciso di utilizzare queste librerie:

- **Requests**: la libreria requests si occupa di ottenere il dataset necessario mediante una richiesta alla pagina web che lo contiene. In questo modo potremo analizzare i dati richiesti.
- **JSON**: questa libreria ci permette di scrivere il dataset ottenuto in precedenza su un file JSON, che verrà poi letto e utilizzato per l'analisi dei dati. In particolare andiamo a utilizzare due funzioni: json.loads(), json.dump() e json.load(). La prima converte le stringhe json in dizionari python, la seconda si occupa di scrivere un dizionario python su un file JSON, la terza si occupa di caricare tutti i record di file in un oggetto python.
- **Sqlite3:** è una libreria software scritta in linguaggio C che ci permette di creare un database in cui successivamente andremo a creare delle tabelle dove immagazzinare i dati raccolti dal dataset. Tutto questo sarà effettuato tramite delle "query" ovvero delle interrogazioni al database. In particolare useremo una CREATE TABLE (DDL) e una INSERT INTO (DML).
- **Matplotlib:** questa libreria si occupa della rappresentazione grafica dei dati da noi richiesti. In particolare nel nostro programma utilizziamo un grafico a linee, anche chiamato grafico storico.

Descrizione del programma

Raccolta dei dati con requests e JSON

Come prima cosa in assoluto il programma si assicura di avere la disponibilità dei dati scaricandoli da una pagina web tramite la funzione “raccoltaDati()” e alcune istruzioni delle librerie “requests” e “JSON”.

In questo modo avremo un dataset aggiornato all’ultima versione in un file JSON in caso l’utente sia online, o in caso contrario aggiornato all’ultimo giorno in cui è stato possibile connettersi alla rete internet.

Dunque le analisi dei dati potranno essere effettuate anche in caso di assenza di connessione.

Inizializzazione del menù principale

All’utente viene presentato un comodo menù in cui può selezionare la funzione da utilizzare, ognuna di queste è collegata ad un numero. Nel caso l’utente inserisca erroneamente un numero non incluso tra quelli indicizzati verrà avvisato con un messaggio e comparirà una richiesta di inserimento di un numero valido.

Dopo di ciò comparirà la lista di tutti gli osservabili su cui sono calcolabili delle statistiche presenti nel dataset.

Inserimento dell’osservabile, date e controlli vari

L’utente dovrà inserire l’osservabile che intende analizzare. Anche qui sarà effettuato un controllo dal programma per assicurarsi che l’input corrisponda ad un osservabile realmente presente nel dataset; questo avviene confrontando quanto immesso con gli elementi una lista apposita, se tutto va a buon fine l’utente potrà procedere nel programma, altrimenti verrà chiesta nuovamente l’immissione di un input valido.

Lo stesso discorso vale per le date di inizio e fine che vengono richieste dal programma subito dopo l'osservabile. Anch'esse subiscono una verifica di correttezza confrontandole con la prima e l'ultima data disponibile nel dataset. Se quanto immesso è compreso o uguale alle date di inizio e fine si potrà procedere con la visione dell'output, in caso contrario il programma richiederà l'immissione di date valide.

Nel caso l'utente inserisca solo anno e mese il programma prenderà automaticamente come giorno il primo del mese.

Importante sottolineare che vengono sempre comunicate all'utente tramite stringhe formattate la prima e l'ultima data disponibile.

Codice commentato

Dall'inizio alla fine il programma è stato commentato con cura da ogni membro del team in modo da massimizzare la leggibilità e la comprensibilità del codice. Per tale scopo ci siamo serviti dei commenti tramite il simbolo "#".

Output di fine esecuzione

Per avvisare l'utente alla fine del programma abbiamo creato due variabili apposite, la prima è chiamata "fine" e contiene il messaggio "Dati calcolati con successo", questa interviene dove l'esecuzione del programma è andata a buon fine.

Mentre la seconda "secondaFine" contiene il messaggio "Fine" e interviene nel caso in cui durante l'esecuzione del programma si sia presentato un imprevisto.

Media Mobile

La funzione **“media mobile”** ha come scopo quello di calcolare la media mobile di una serie di ***dati*** in un periodo assegnato. Per fare ciò, forniamo 5 parametri formali:

1. Dataset: rappresenta il dataset da cui estrarre i dati e su cui calcolare la media mobile.

2. Data_inizio: indica l’inizio del periodo di tempo in cui calcolare la media mobile.

3. Data_fine: indica la fine del periodo di tempo in cui calcolare la media mobile.

4. Size: indica il numero di elementi su cui calcolare la media mobile.

5. Osservabile: rappresenta l’osservabile di cui si vogliono conoscere i valori.

La funzione inizia con l’estrazione dei dati presenti nel dataset, filtrando solo quelli compresi nell’intervallo di tempo che abbiamo specificato e che hanno un valore per l’osservabile sempre da noi richiesto.

Successivamente, abbiamo utilizzato due cicli for per calcolare la media mobile di size elementi spostandosi di un elemento alla volta nella lista dei dati estrapolati.

Per finire, il programma controllerà se la lista dei risultati è vuota e, in caso affermativo, stamperà un messaggio di avviso all’utente che sta interagendo con il programma, altrimenti se la lista dei risultati non risulta essere vuota esso restituirà la lista con al suo interno i risultati.

Spiegazione del codice:

1. Per cominciare abbiamo definito la funzione **media mobile**: def mediaMobile(dataset, data_inizio, data_fine, size, osservabile):

2. La funzione inizia dichiarando la lista ***"elem_oss"*** che conterrà gli elementi osservati, ovvero tutti i dati del dataset che abbiamo deciso di filtrare in base all'intervallo di tempo e l'osservabile che abbiamo specificato.

"elem_oss = [ ]" → *comando per la creazione della lista*

3. Abbiamo quindi avviato un ciclo for sui dati del dataset. Per ciascun elemento del dataset, è stata salvata la data in ***"data_agg"*** e si verificherà che il valore dell'osservabile non sia ***"null"*** e che la data sia compresa nell'intervallo di tempo che abbiamo specificato. In caso positivo il valore dell'elemento osservato verrà aggiunto alla lista ***"elem_oss"***.

```
" for i in datiCovid:

    data_agg = i["data"][:10]

        if i[osservabile] is not None and data_agg >= data_inizio and data_agg <= data_fine:

            elem_oss.append(i[osservabile]) "
```

4.E' stata quindi dichiarata una lista ***"result"*** che conterrà i risultati delle medie mobili che sono state calcolate.

```
" result = [ ] "
```

5.Abbiamo nuovamente eseguito un altro ciclo for sulla lista ***"elem_oss"***, che comprenderà tutti gli elementi tranne l' ultimo elemento **size** (size è il numero di elementi su cui si vuole calcolare la media).

```
" for i in range(0, len(elem_oss) - size):
```

```
sum = 0 "
```

6.All'interno del ciclo for che è stato precedentemente creato viene avviato un altro ciclo for sugli ultimi size elementi della lista ***"elem_oss"***, a partire dall'elemento corrente del precedente ciclo for. Per ogni elemento dell'ultimo ciclo creato verrà sommata al totale ***"sum"***.

```
" for j in range(0, size-1):

        sum += elem_oss[i + j] "
```

7.Al termine del ciclo for interno, verrà calcolata la media mobile avendo specificato al programma di eseguire la divisione tra somma e size. Una volta che terminerà il calcolo da noi richiesto aggiungerà il risultato alla lista ***"result"*** avendo usato la funzione append.

```
" result.append(sum / size) "
```

8.Una volta terminata l'operazione, se il programma riuscirà ad ottenere dei dati da noi richiesti, ci verranno mostrati in output.

```
" if len(result)==0:

    print("Numero minimo valori per calcolare la media mobile non raggiunto")

    return secondaFine

  else:

return result " .
```

Funzione Grafico

La funzione grafico si occupa di rappresentare graficamente l'andamento di un'osservabile in uno specifico intervallo temporale. In questo modo si possono osservare i picchi, le valli, gli aumenti e i crolli o la monotonia dei valori di un'osservabile aderente all'ambito covid.

Si basa su 4 parametri formali, dataset, ovvero l'insieme dei valori delle osservabili contenute nel file JSON, data_inizio, ovvero la data a partire dalla quale iniziare a considerare i valori dell'osservabile, data_fine, ovvero l'ultima data di cui analizziamo l'osservabile, e l'osservabile, ovvero una grandezza misurabile, nostro oggetto di studio.

L'utente, scegliendo la funzione grafico nel menù, deve fornire 3 valori: una data iniziale, una finale e l'osservabile a cui è interessato.

La funzione grafico genera inoltre un file in formato Database, nel quale inserisce solamente le date comprese fra quella iniziale e quella finale scelta dall'utente con i relativi valori dell'osservabile, sempre scelta dall'utente. In questo modo si possono eseguire diverse operazioni di analisi dei dati tramite linguaggio SQL, ovvero con applicazioni come SQLiteStudio.

Spiegazione del codice step by step:

Una volta definito il nome della funzione e i parametri formali, procediamo a creare due liste vuote, ovvero *elem_oss* e *data_lista* . Al loro interno andremo ad inserire rispettivamente i valori dell'osservabile e le date comprese fra *data_inizio* e *data_fine*.

Creiamo un ciclo for, attraverso il quale, per ogni elemento contenuto nella lista datiCovid (lista composta da tanti dizionari quanti sono i giorni totali considerati nel

link Github, e quindi nel File JSON da noi salvato su disco per accedervi anche offline), innanzitutto modifichiamo le date estratte da ogni dizionario, sfruttando la loro struttura di "chiave"-"valore", ovvero facendo riferimento alla chiave denominata "data", e riducendo il valore ai primi 10 caratteri , ovvero rimuovendo la parte di stringa relativa all'orario in cui sono stati raccolti quei dati (in quanto le stringhe sono considerate da python come oggetti iterabili e quindi ogni loro carattere è indicizzabile). Questo valore aggiornato lo assegniamo a una nuova variabile che chiamiamo *data_agg*.

A questo punto creiamo una selezione con il token *if*. La condizione è che la *data_agg* sia compresa fra la data d'inizio e la data di fine. Se la condizione è soddisfatta, la funzione inserisce nelle due liste, tramite il metodo "append", rispettivamente i valori filtrati dell'osservabile e le date comprese fra inizio e fine. Queste due liste ci serviranno in seguito per costruire il nostro grafico.

Procediamo quindi modificando alcuni caratteri nelle variabili data_inizio, data_fine e osservabile, sempre tramite il metodo append, per renderle più gradevoli alla vista una volta inserite nel grafico.

Tramite la libreria matplotlib, che abbiamo importato all'inizio, costruiamo il grafico in questo modo: con il metodo "figure" definiamo la grandezza dell'immagine, con "title" definiamo il titolo del grafico con una stringa formattata nella quale inseriamo: data iniziale, finale e osservabile. Con il metodo "xticks" ruotiamo i valori sull'asse x di 60 gradi, con il metodo "plot" creiamo l'effettivo grafico fornendogli come dati i valori contenuti nelle liste precedentemente create e impostiamo il colore della linea su rosso, con il metodo "legend" creiamo una leggenda per il grafico, e infine con il metodo "grid" inseriamo una griglia orizzontale in sovrimpressione. Se ci limitiamo a queste righe di codice, nel caso in cui le date di inizio e fine siano molto distanti, i valori rappresentati sarebbero molto numerosi, e i valori sull'asse x diventerebbero illeggibili. Per questo introduciamo altre linee di codice sfruttando ulteriori metodi della libreria matplotlib.

Tramite il metodo “gca” troviamo le posizioni esatte di ogni indice sull'asse X, poi tramite il metodo “get_ticklocs” otteniamo le locazioni di ogni indice, indicando sempre che l'operazione andrà svolta sull'asse X. Procediamo a impostare un numero di indici sull'asse X che consenta le leggibilità ottimale. Eseguiamo una divisione parte intera (floor division), in cui otteniamo il risultato arrotondato per difetto all'intero più vicino. Andiamo quindi a definire gli indici che verranno mostrati sul grafico. Infine tramite il metodo “set_ticks” impostiamo i nuovi indici che saranno presenti sul grafico.

A questo punto con il metodo “savefig” salviamo il grafico sul disco e lo mostriamo con il metodo “show”.

Stampiamo quindi la stringa: grafico salvato con successo

Procediamo ora con la creazione del database.

Creiamo una nuova lista chiamata *data_lista_agg*, nella quale inseriremo le date modificate tramite il ciclo *for* successivo, per riportarle nel database in un formato più comunemente utilizzato, ovvero con anno mese e giorno separati da slash.

Facciamo lo stesso con la nuova lista vuota chiamata *elem_sqlite*, in cui inseriremo sempre tramite ciclo *for* tutti i valori None con valori NULL, in quando il formato SQL non accetta valori None (utilizza come corrispettivo il valore “NULL”).

Apportiamo delle modifiche alle date e all’osservabile che avevamo precedentemente modificato per renderle utilizzabili con i database (lo slash viene interpretato come divisione da sqlite quindi dobbiamo sostituirlo con il trattino basso).

A questo punto creiamo il nome della tabella, sempre tramite stringa formattata, che conterrà i valori osservati e le relative date .

Tramite la libreria “sqlite3” e il metodo “connect” ci colleghiamo al database. Questo avrà come nome la stringa formattata precedente. Collegarsi al database vuol dire che, nel caso in cui non esistesse già, viene creato un file in formato database, sul quale eseguiremo le operazioni successive.

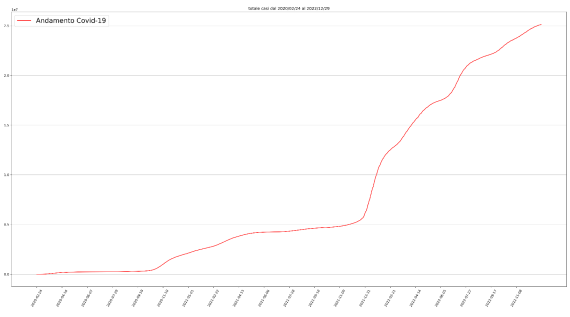
Creiamo a questo punto una query, ovvero un comando SQL da eseguire attraverso il metodo “execute”. Questa query si occuperà di creare una tabella, se già non esiste, all’interno del nostro database, i cui attributi saranno le date e l’osservabile, e la loro tipologia rispettivamente testuale e reale.

Tramite il metodo “commit” salviamo i cambiamenti sul disco.

Costruiamo un altro ciclo *for* che ci servirà per l’inserimento dei dati nella nostra tabella

Utilizziamo la funzione zip, che aggrega gli elementi da due o più iterabili, per unire gli elementi della lista *data_lista_agg* e *elem_sqlite*. Trasformiamo ogni elemento della prima lista in una stringa tramite l’aggiunta di apici. Scriviamo quindi la query per l’inserimento dei dati sotto forma di stringa formattata. Nella tabella verranno inseriti per ogni ciclo, nella colonna della data e in quella dell’osservabile i rispettivi valori, il primo testuale e il secondo reale.

Concludiamo eseguendo la query e salvando i cambiamenti sul disco



esempio di un risultato ottenuto inserendo come osservabile desiderato il totale dei casi e come date d’inizio 2020-02-24 e 2022-12-29

Funzione Somma Cumulata

La funzione ***somma cumulata*** fornisce una lista contenente la somma cumulata di una determinata osservabile. Questa funzione prende ***in input 4 parametri*** che sono rispettivamente il ***dataset***, ovvero i dati contenuti nel file JSON, la ***data di inizio***, da dove iniziamo a considerare l' osservabile, la ***data di fine***, ovvero la data fino alla quale analizziamo l'osservabile richiesta ed infine ***l' osservabile*** da analizzare.

Questa funzione può essere scelta tramite il ***menu di selezione*** che abbiamo creato, e per utilizzarla bisogna effettuare i seguenti procedimenti:

1 | Il primo step, dopo aver avviato il programma, è quello di selezionare dal menù la ***funzione numero 3***, ovvero quella della somma cumulata.

2 | Dopo averla selezionata, il programma ci chiederà di fornire degli input, che in ordine sono: l' osservabile che si desidera analizzare, ***ad esempio gli ingressi in terapia intensiva***, poi la data di inizio da cui si vuole far partire la somma cumulata, ed infine la data di fine fino alla quale si vuole ottenere la lista.

3 | L' output che ci verrà fornito dalla funzione sarà quindi una lista con come primo valore il valore effettivo dell' osservabile nel primo giorno, mentre nei giorni successivi i valori saranno cumulati tra di loro.

Spiegazione del codice step by step:

1| Definiamo la funzione: def sommatoria(dataset, data_inizio, data_fine, osservabile)

2| Digitando il comando ***help(sommatoria)*** ci verrà fornita una breve guida sull'utilizzo della funzione

3| Procediamo nel creare la funzione effettiva, andando prima a creare ***due liste vuote***, rispettivamente ***data_lista*** e ***result***, dove una conterrà la data mentre l'altra il risultato.

4| Creo una lista chiamata ***osservabili_cumulate,*** dove vengono inserite le osservabili che risultano ***già cumulate*** nel dataset, in modo che all'avvio della funzione, se l'osservabile scelta fosse una all'interno della lista delle osservabili già cumulate, ritornerà una lista non cumulata, al contrario, se l'osservabile non è presente nella lista delle osservabili già cumulate, la funzione ci ritornerà una lista di somma cumulata per l'osservabile che abbiamo scelto.

5| La variabile somma_cumulata partirà da 0, di modo che i valori successivi verranno "caricati" al suo interno "appendendoci" la variabile result.

6| Creiamo un ***ciclo for***, prendendo in considerazione i datiCovid, ovvero i dati che in precedenza abbiamo caricato *(vedere la parte comune)*, dove sono contenute le coppie chiave-valore che ci interessano.

7| Definiamo la condizione per la quale la data sia compresa tra quella di inizio e quella di fine.

8| Viene usata la condizione ***if … is not None …*** di modo che i valori mancanti del dataset vengano saltati, evitando che il programma ci dia un errore.

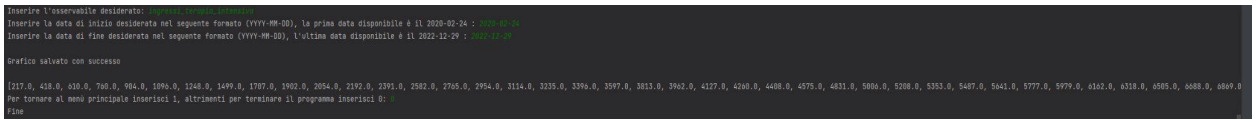
9| Usiamo l' operatore logico ***and*** per fare in modo che le condizioni da verificare siano entrambe.

10| Andiamo ad aggiungere a somma_cumulata i valori dell'osservabile, in modo da cumularli.

11| Infine con ***return result*** diciamo alla funzione che il nostro output dovrà essere la lista contenente la somma cumulata dell' osservabile.

Esempio della funzione sommatoria:

Input → ingressi in terapia intensiva dal 2020-02-24 al 2022-12-29



Spiegazione del codice del grafico della somma cumulata:

In questo caso grazie alla libreria ***matplotlib.pyplot*** che abbiamo precedentemente importato, possiamo scrivere il codice per effettuare il grafico della funzione sommatoria:

1| Definiamo inizialmente la figura del grafico

2| Tramite una stringa formattata permettiamo che venga inserito il titolo del grafico, dove viene mostrata l' osservabile selezionata e le date.

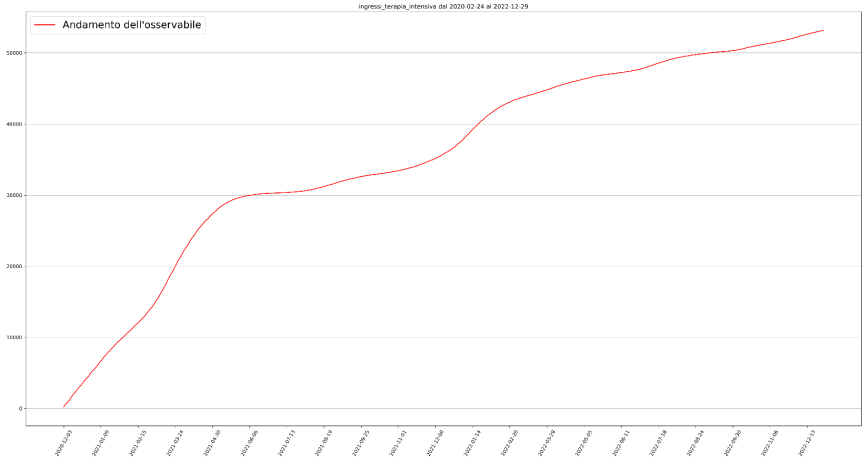
3| Definiamo la rotazione che avranno le date sull' asse X.

4| Diciamo alla funzione cosa graficare, ovvero la data scelta in precedenza e il risultato della lista contenente la somma cumulata dell'osservabile scelta.

5| Tramite il comando ***plt.savefig*** possiamo automaticamente salvare il grafico che abbiamo appena creato con una funzione di immagine di 300 DPI.

Esempio dell'output fornito dal codice del grafico della somma cumulata:

Input → ingressi in terapia intensiva dal 2020-02-24 al 2022-12-29



Osservando l' immagine notiamo che nonostante sia un grafico di somme cumulate, non cresce sempre allo stesso modo, mostrando come in un determinato periodo siano diminuiti gli ingressi in terapia intensiva.
N.B: L'osservabile presa in considerazione è quella degli ingressi in terapia intensiva.

Funzione Valore Massimo

La funzione valore massimo ha l'obiettivo di mostrare il valore più alto tra quelli presenti nel dataset, per ogni elemento della lista: (listaOsservabili); in un periodo indicato dall'utente.

La funzione è stata creata puntando ad una massima ottimizzazione del progetto, per avere maggior chiarezza ed affidabilità sull'esecuzione della funzione.

Inizialmente la funzione estrae dal dataset i primi 10 caratteri della stringa in input, ovvero l'anno, il mese ed il giorno visualizzato, non considerando l'orario seguente alla data.

In seguito abbiamo creato due liste vuote:

- listaOsservabili -> contenente i valori raccolti dal osservabile richiesta
- data -> contenente le date relative a ciascun osservabile

Abbiamo utilizzato vari cicli per rendere più efficiente la funzione; in particolare abbiamo utilizzato un ciclo for per raccogliere i caratteri della data di ciascun osservabile ed inserirli nella "listaOsservabili", di conseguenza la data viene automaticamente inserita nella lista "data".

Infine ci siamo serviti di un costrutto "try-except" per eseguire le funzioni del programma ed in caso la lista degli osservabili sia vuota il programma restituirà un messaggio di avviso, terminando l'esecuzione. Abbiamo inserito un ciclo for in modo da definire il valore massimo e la relativa data del dataset per ogni osservabile. Così facendo troveremo l'elemento più grande nella lista e subito dopo troviamo automaticamente anche la relativa data.

Finiti i calcoli vengono stampati tramite una stringa formattata il valore massimo e la rispettiva data. In caso non venga trovato nessun valore nell'arco di tempo definito dall'utente, interverrà la clausola "except" del rispettivo costrutto definendo la variabile massimo come una stringa contenente il messaggio: "Nessun valore registrato". In aggiunta verrà stampato un messaggio di avviso per l'utente, indicando che non è stato possibile trovare il massimo per assenza di dati. ed infine tramite il "return" verrà stampato un messaggio che avvisa l'utente della corretta esecuzione del programma e della sua conclusione.

```
Inserire l'osservabile desiderato: casi_testati
Inserire la data di inizio desiderata nel seguente formato (YYYY-MM-DD), la prima data disponibile è il 2020-02-24 : 2020-02-24
Inserire la data di fine desiderata nel seguente formato (YYYY-MM-DD), l'ultima data disponibile è il 2022-12-29 : 2022-12-29


Il massimo è: 66823327.0 e ha data: 2022-12-29

Dati calcolati con successo
```

esempio di un risultato ottenuto inserendo come osservabile desiderato casi testati e come date d'inizio 2020-02-24 e 2022-12-29. In questo caso il programma mostra come il massimo è presente in data 2022-12-29 e vale 66823327.0

Funzione Valore Minimo

La funzione **valoreMinimo** ha come obiettivo il **trovare il valore minimo** di **una qualunque osservabile** in un **periodo di tempo assegnato** dall'utente.

Dispone di un help richiamabile dall'utente in caso di dubbi sul funzionamento del metodo o sull'uso dei parametri di quest'ultimo.

Come nome abbiamo optato per valoreMinimo e come parametri formali:

- **dataset**: il dataset letto dal file JSON creato dalla funzione raccoltaDati() e alcune istruzioni della libreria JSON.
- **dataInizio**: il secondo parametro formale non è altro che la data d'inizio del periodo in cui l'utente vuole eseguire un'analisi per trovare il valore minimo. Questa viene indicata subito dopo aver inserito l'osservabile interessato.
- **dataFine**: il terzo parametro formale rispecchia il secondo, si tratta infatti della data di fine del periodo in cui l'utente vuole eseguire l'analisi. Questa viene indicata subito dopo aver inserito la data d'inizio.
- **osservabile**: il quarto parametro formale è il valore dell'osservabile su cui effettivamente si andrà ad eseguire l'analisi dei valori per trovare il minimo. Questo viene inserito dall'utente durante l'utilizzo del programma.

Dopo che l'utente ha finito di compilare i campi richiesti il dataset viene passato ad un ciclo for che controllando se la data letta dal record è inclusa nel periodo di tempo assegnato scarta o scrive il valore dell'osservabile e la relativa data in due apposite liste.

Dopo aver fatto **un'accurata selezione dei dati mantenendo esclusivamente i dati non "null"**, quindi solo quelli su cui possono essere calcolate delle statistiche, questi vengono trasformati in dati di tipo "float". A questo punto le liste contenenti i valori dell'osservabile e le date di questi ultimi sono pronte per essere analizzate.

Tramite un costrutto "**try-except**" eseguiamo la ricerca del minimo tramite un semplice ciclo for di lunghezza pari a quella della lista contenente i valori dell'osservabile.

Confrontando ogni valore letto dal ciclo con uno di riferimento (che viene aggiornato ogni volta che si trova un elemento minore di quest'ultimo) **troveremo l'elemento più piccolo nella lista** e subito dopo **anche la relativa data**.

Finiti i calcoli vengono stampati tramite una stringa formattata il valore minimo e la rispettiva data.

In seguito la funzione genererà un grafico rappresentante l'andamento dell'osservabile con il minimo segnalato tramite un triangolino giallo all'interno e rosso ai bordi lungo la linea tracciata.

Dopo la rappresentazione grafica il programma provvederà a salvare come file in formato ".png" con un DPI (Punti per pollice) = 300 il grafico tracciato.

Se non dovesse essere trovato alcun dato nell'arco di tempo assegnato dall'utente interverrà la clausola "**except**" definendo la variabile minimo come una stringa contenente il messaggio "Nessun valore registrato".

In seguito quest'ultima verrà stampata avvisando l'utente che non è stato possibile trovare il minimo a causa dell'assenza di dati.

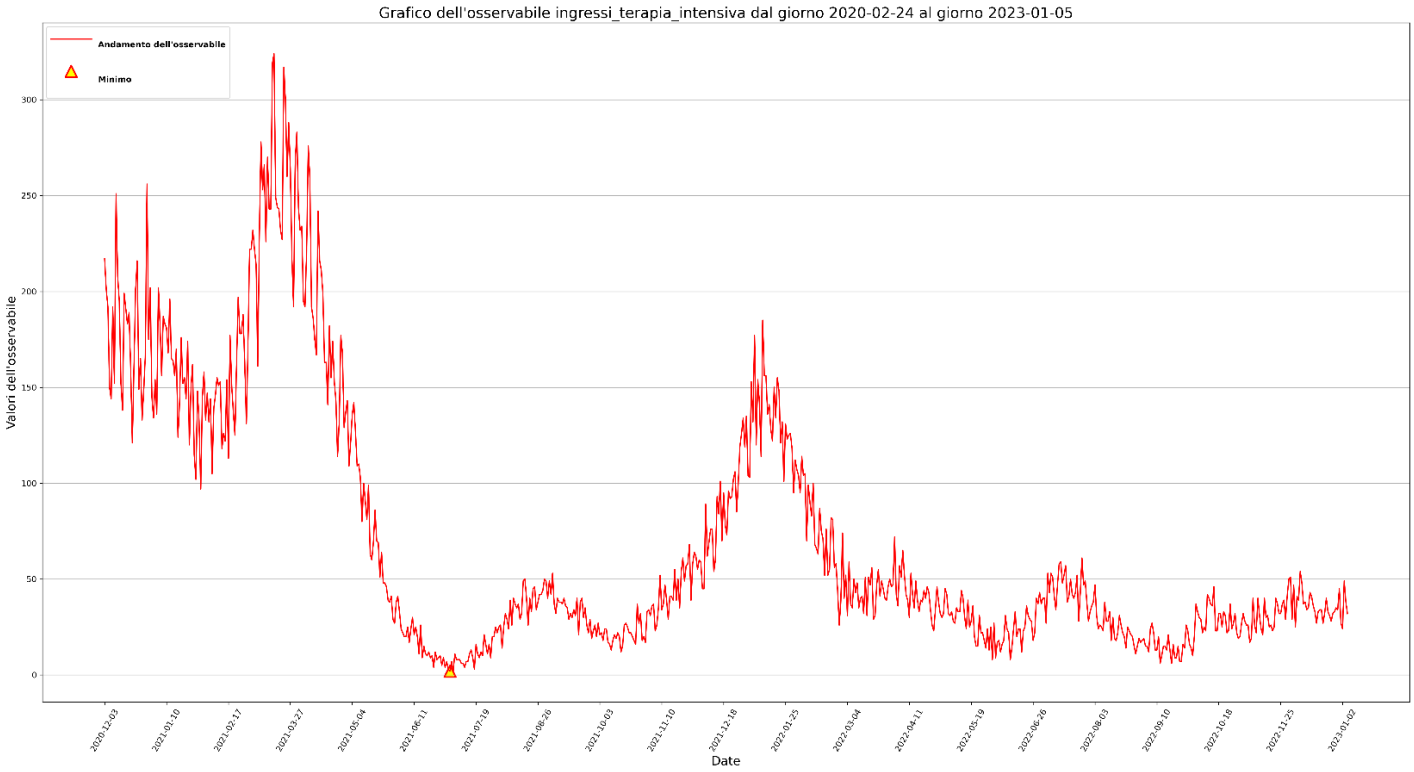
Infine la funzione ritornerà a seconda del risultato tramite una una clausola if-else e delle istruzioni "return" un messaggio che avvisa l'utente della fine dell'esecuzione del programma.

```
Inserire l'osservabile desiderato: casi_testati
Inserire la data di inizio desiderata nel seguente formato (YYYY-MM-DD), la prima data disponibile è il 2020-02-24 : 2020-02-24
Inserire la data di fine desiderata nel seguente formato (YYYY-MM-DD), l'ultima data disponibile è il 2022-12-29 : 2022-12-29


Il minimo è: 935310.0 e ha data: 2020-04-19

Dati calcolati con successo
```

Esempio di un risultato ottenuto inserendo come osservabile desiderato casi_testati e come date d'inizio 2020-02-24 e 2022-12-29. In questo caso il programma mostra come il minimo è presente in data 2020-04-19 e vale 935310.0



Esempio di grafico ottenuto osservando l'andamento degli ingressi in terapia intensiva dal 2020-02-24 al 2023-01-05