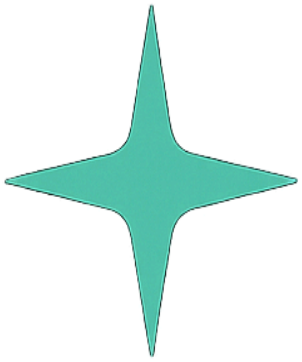
Istituto Istruzione Superiore
"G. Vallauri" - Fossano



PROGETTO APOLLO

Gallo Denislav
Rivarossa Davide
5^C Informatica

Anno scolastico 2024/25

Tesina Esame di Stato

Indice

1. Introduzione

1.1 Contesto del Progetto

La gestione e la comprensione delle dinamiche epidemiche rappresentano una sfida cruciale per la società moderna, come ampiamente dimostrato dalla recente pandemia di COVID-19. La capacità di analizzare dati storici, prevedere scenari futuri e comunicare queste informazioni in modo chiaro ed efficace è fondamentale per supportare decisioni informate a livello sanitario e politico. Il presente progetto, denominato "Apollo", si inserisce in questo contesto come elaborato per l'Esame di Stato, con l'intento di esplorare e implementare soluzioni tecnologiche avanzate per l'analisi epidemica.

1.2 Obiettivi e Motivazioni

L'obiettivo primario del progetto Apollo è sviluppare una piattaforma software prototipale, concepita come applicazione web full-stack, per l'analisi, la previsione e la visualizzazione interattiva dell'andamento di epidemie. La motivazione principale risiede nella volontà di superare le tradizionali modalità di reporting statico, spesso limitate a grafici e tabelle predefinite, offrendo invece uno strumento dinamico che integri modellistica predittiva basata su Machine Learning con visualizzazioni dati avanzate, inclusa una rappresentazione geospaziale 3D. Si è scelto di utilizzare come caso di studio i dati storici nazionali del COVID-19 in Italia, data la loro disponibilità e rilevanza.

---

2. Il Progetto Apollo: Panoramica Generale

2.1 Il Problema: Limiti del Reporting Statico

Durante le emergenze sanitarie, la divulgazione dei dati sull'andamento epidemico avviene spesso attraverso report statici (PDF, immagini di grafici, tabelle fisse). Sebbene utili per una visione d'insieme, questi strumenti presentano limitazioni significative:

- **Scarsa Interattività:** L'utente non può esplorare i dati dinamicamente, filtrare informazioni o personalizzare le visualizzazioni.
- **Aggiornamenti Ritardati:** La produzione di report statici richiede tempo, portando a possibili ritardi nella disponibilità delle informazioni più recenti.
- **Difficoltà di Integrazione Predittiva:** L'inclusione di modelli predittivi e la visualizzazione delle loro incertezze risultano complesse.
- **Visualizzazioni Limitate:** Spesso mancano rappresentazioni geospaziali intuitive o la possibilità di analizzare diverse sfaccettature dei dati contemporaneamente.

2.2 La Soluzione: Apollo, una Piattaforma Dinamica

Apollo nasce con l'ambizione di indirizzare queste limitazioni. Si tratta di una piattaforma web prototipale che mira a:

- **Centralizzare i dati epidemici** e renderli accessibili tramite un'interfaccia intuitiva.
- **Integrare modelli di Machine Learning** per la previsione a breve termine dell'evoluzione dell'epidemia.
- **Offrire visualizzazioni dati dinamiche e interattive**, sia in formato 2D (grafici temporali) che 3D (mappamondo).
- **Fornire strumenti per analizzare le componenti del modello** e comprendere meglio i fattori che influenzano le previsioni.

L'idea è quella di fornire uno strumento che non sia solo un visualizzatore di dati passati, ma un vero e proprio laboratorio di analisi e previsione.

2.3 Caso di Studio: Analisi dei Dati COVID-19 in Italia

Per concretizzare e testare le funzionalità di Apollo, è stato scelto come caso di studio l'andamento nazionale della pandemia di COVID-19 in Italia. I dati utilizzati provengono dal repository ufficiale della Protezione Civile italiana (file *dpc-covid19-ita-andamento-nazionale.csv*), che fornisce serie storiche giornaliere su diverse metriche chiave (nuovi positivi, deceduti, ricoverati, ecc.). Questa scelta permette di lavorare con dati reali, complessi e di grande impatto, validando l'efficacia degli strumenti sviluppati.

---

3. Architettura della Piattaforma

Apollo è stato progettato seguendo un'architettura full-stack moderna, con una netta separazione tra la logica di backend (gestione dati, analisi, previsioni) e l'interfaccia utente frontend (visualizzazione, interazione).

3.1 Architettura Full-Stack: Backend e Frontend

Questa separazione offre numerosi vantaggi, tra cui maggiore modularità, scalabilità e manutenibilità. Il backend espone i dati e le funzionalità attraverso API (Application Programming Interface), mentre il frontend consuma queste API per presentare le informazioni all'utente.

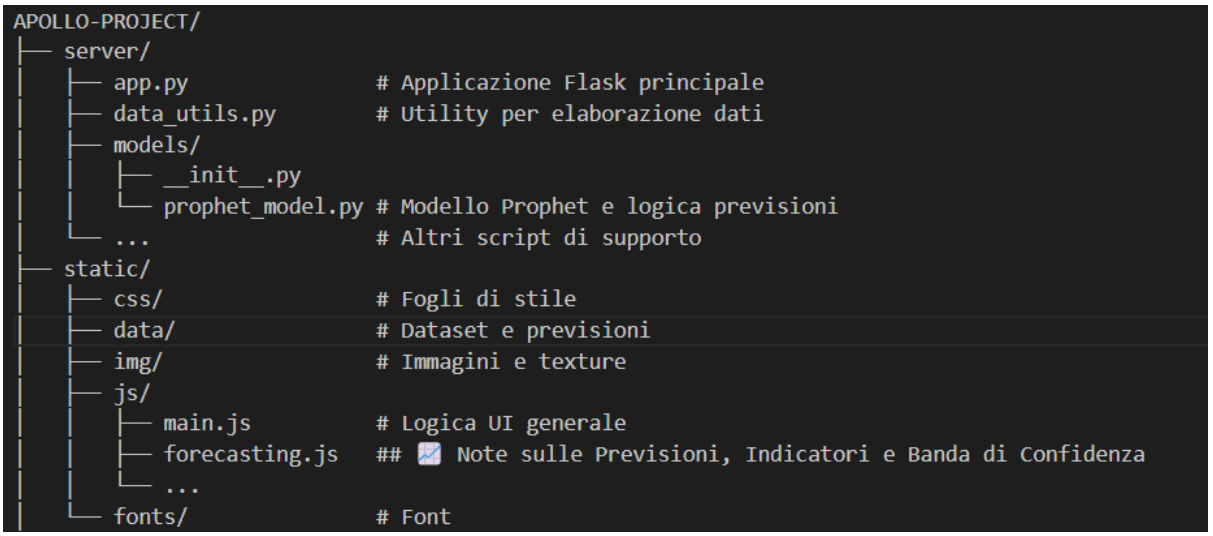


immagine relativa all’architettura del progetto con separazione tra Backend e Frontend

3.2 Backend (Python/Flask)

Il backend è il motore di Apollo, responsabile dell'elaborazione dei dati e della generazione delle previsioni. È stato sviluppato in Python, sfruttando il microframework Flask.

3.2.1 Microframework Flask e API RESTful

Flask è stato scelto per la sua leggerezza, flessibilità e la vasta comunità di supporto. Permette di creare rapidamente applicazioni web e API. In Apollo, Flask è utilizzato per costruire una serie di **API RESTful** (Representational State Transfer). Queste API, documentate nel file *DOCUMENTAZIONE_API_DEBUG_BACKEND.md*, agiscono come "ponti" che permettono al frontend di richiedere e ricevere dati (dati storici, previsioni, dati per il globo, metriche di performance) in un formato standardizzato, tipicamente JSON (JavaScript Object Notation). Il file principale del server è *server/app.py*.

3.2.2 Modulo di Analisi e Previsione Dati

La logica di analisi e previsione è incapsulata in moduli Python dedicati, principalmente *server/data_utils.py* per la preparazione e manipolazione dei dati, e *server/models/prophet_model.py* che contiene la logica specifica del modello predittivo. Questa modularizzazione rende il codice più organizzato e facile da aggiornare.

3.2.3 Il Cuore Predittivo: Il Modello Prophet

Per le previsioni dell'andamento epidemico (ad esempio, la stima dei *nuovi_positivi*), Apollo utilizza **Prophet**, una libreria open-source sviluppata da Meta (Facebook) specificamente progettata per la previsione di serie temporali. Prophet è robusto di fronte a dati mancanti, shift nel trend e gestisce bene la stagionalità e gli effetti delle festività. Nel progetto, il modello Prophet è configurato con parametri ottimizzati identificati durante la fase di sviluppo, come:

- **Stagionalità moltiplicativa:** Adatta per serie temporali la cui stagionalità varia in proporzione al livello del trend.
- **Festività italiane:** Per tenere conto dell'impatto di specifiche festività nazionali dell'andamento epidemico.
- `changepoint_prior_scale=0.5`: Un parametro che regola la flessibilità del modello nell'identificare i punti di cambiamento nel trend. Il file *server/train_prophet_models.py* contiene script per l'addestramento (o ri-addestramento) dei modelli Prophet, mentre *server/prophet_train_and_forecast.py* gestisce il flusso completo di addestramento e generazione di previsioni.

3.2.4 Gestione dei Dati con Pandas e Valutazione con Scikit-learn

La libreria **Pandas** è ampiamente utilizzata nel backend per caricare, pulire, trasformare e analizzare i dati tabellari delle serie storiche (provenienti dai file *.csv*). Per valutare l'accuratezza delle previsioni del modello Prophet, vengono calcolate diverse metriche di errore utilizzando la libreria **Scikit-learn**. Le metriche principali includono:

- **MAE (Mean Absolute Error):** Errore medio assoluto.
- **RMSE (Root Mean Squared Error):** Radice quadrata dell'errore quadratico medio, penalizza maggiormente gli errori grandi.
- **MAPE (Mean Absolute Percentage Error):** Errore percentuale medio assoluto, utile per confrontare l'accuratezza su scale diverse.

3.3 Frontend (HTML, CSS, JavaScript)

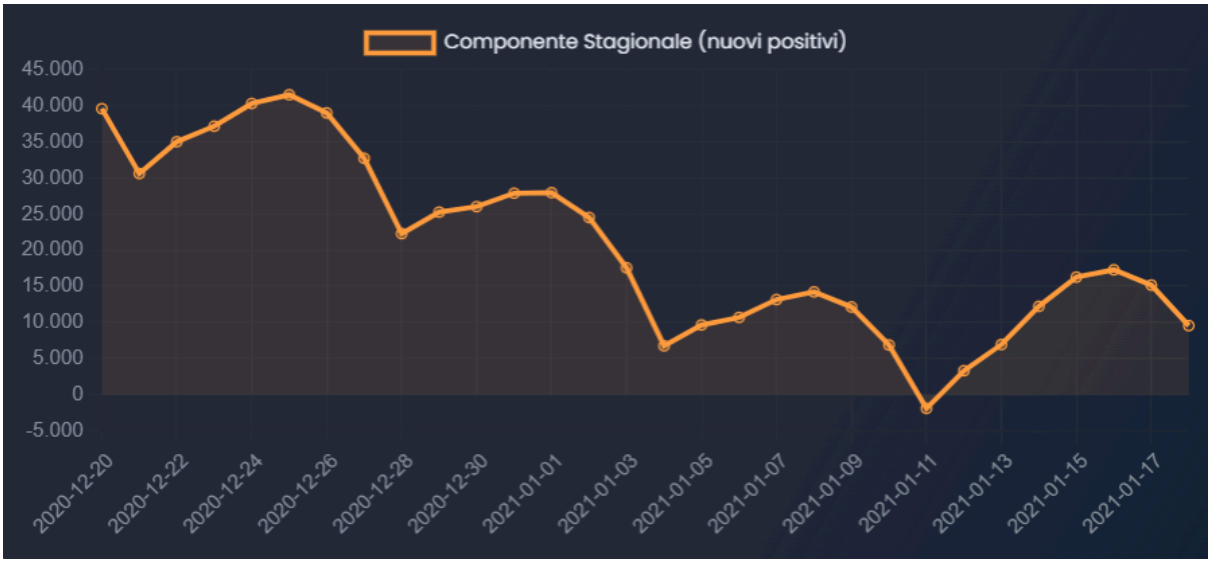
Il frontend è l'interfaccia con cui l'utente interagisce. È costruito utilizzando tecnologie web standard: HTML per la struttura, CSS per lo stile e JavaScript per la logica lato client e l'interattività.

3.3.1 Struttura e Styling dell'Interfaccia

Le pagine web sono definite tramite file **HTML**, utilizzando il sistema di templating **Jinja2** integrato in Flask (file nella cartella *templates/*, come *landing.html*, *previsioni.html*, *statistiche.html*, *informazioni.html*). Jinja2 permette di inserire dinamicamente dati provenienti dal backend all'interno delle pagine HTML. Lo **CSS** (fogli di stile in *static/css/*) è utilizzato per definire l'aspetto grafico dell'applicazione, garantendo uno styling moderno, personalizzato e responsivo, adattabile a diversi dispositivi.

3.3.2 Visualizzazioni 2D con Chart.js

Per la rappresentazione di grafici temporali standard (come l'andamento storico dei casi, il confronto tra dati reali e previsioni, le componenti del modello Prophet come trend e stagionalità), Apollo integra la libreria JavaScript **Chart.js**. La logica per la creazione e l'aggiornamento di questi grafici è gestita principalmente nei file *static/js/statistics.js* e *static/js/forecasting.js*. Chart.js è scelta per la sua semplicità d'uso e la buona qualità dei grafici prodotti.



Ecco un esempio di grafico con Chart.js

3.3.3 Visualizzazione Geospaziale 3D con Three.js

Una delle caratteristiche distintive di Apollo è la visualizzazione interattiva dei dati su un mappamondo 3D. Questa funzionalità è implementata utilizzando la potente libreria JavaScript **Three.js**, specializzata nella creazione e visualizzazione di grafica 3D nel browser. I file chiave per questa componente sono:

- *static/js/globe.js*: Logica principale per la creazione della scena 3D, il caricamento del globo e la gestione delle interazioni.

- *static/src/threeGeoJSON.js*: Utility per convertire dati GeoJSON in geometrie 3D per Three.js.
- *static/src/getStarfield.js*: Per creare uno sfondo stellato, migliorando l'estetica della visualizzazione.
- *static/js/OrbitControls.js*: Permette all'utente di navigare la scena 3D (ruotare, zoomare, pan).

La visualizzazione include:

- Una **sfera terrestre texturizzata** utilizzando dati **GeoJSON** (come *ne_110m_land.json*) per rappresentare le masse terrestri.
- La capacità di **convertire coordinate geografiche** (Latitudine/Longitudine) in coordinate cartesiane 3D per posizionare oggetti o indicatori sulla superficie del globo, basati sui dati forniti dall'API backend (es. `/api/data/globe`).

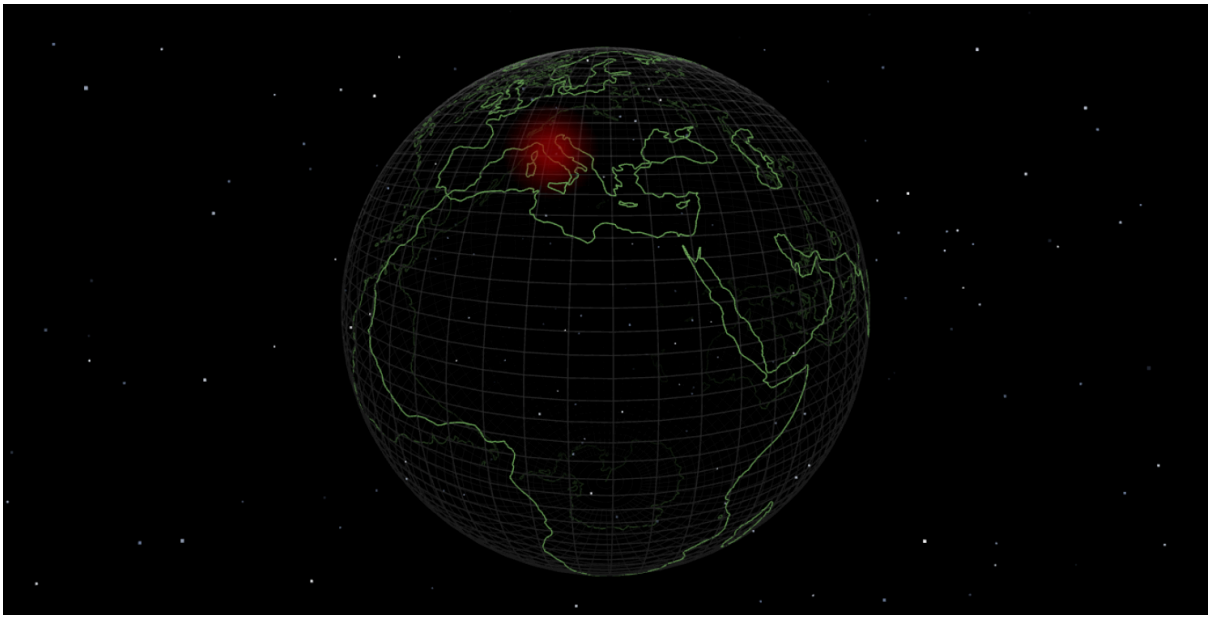


Immagine del mappamondo in 3D con indicatori sulla superficie

4.Tecnologie chiave da implementare

Per comprendere a fondo il funzionamento e lo sviluppo di Apollo, è fondamentale analizzare le tecnologie principali che ne costituiscono la base e che ne permettono il funzionamento.

4.1 Python e il suo ecosistema (Flask, Pandas, Scikit-learn)

Python è uno dei linguaggi di programmazione più diffusi e apprezzati, soprattutto nel campo della scienza dei dati, dello sviluppo web e dell’intelligenza artificiale. La sua sintassi semplice e leggibile lo rende accessibile anche ai principianti, ma allo stesso tempo è estremamente potente grazie al vasto ecosistema di librerie disponibili.

Nel contesto di questo progetto, Python è stato scelto per la sua versatilità e per la disponibilità di strumenti altamente specializzati, tra cui:

- **Flask**: un micro-framework per lo sviluppo web che consente di creare applicazioni leggere ma estensibili. È stato utilizzato per sviluppare la parte server-side del progetto, gestendo le richieste HTTP, i percorsi (routes) e la comunicazione tra frontend e backend.

- **Pandas**: una libreria essenziale per la manipolazione e l’analisi dei dati. Pandas permette di gestire tabelle, serie temporali e insiemi di dati in modo efficiente e intuitivo, facilitando operazioni come il filtraggio, l’aggregazione e la trasformazione dei dati.

- **Scikit-learn**: una delle librerie più utilizzate per il machine learning in Python. Contiene numerosi algoritmi per l’apprendimento supervisionato e non supervisionato, oltre a strumenti per la valutazione dei modelli e la selezione delle caratteristiche. In questo progetto, Scikit-learn è stato impiegato per [inserisci brevemente l’obiettivo: ad esempio, classificare dati, effettuare previsioni, ecc.].

L’utilizzo combinato di queste tecnologie ha permesso di sviluppare un’applicazione completa, capace non solo di interfacciarsi con l’utente tramite il web, ma anche di elaborare e analizzare dati in modo efficiente.

4.2 Prophet di Meta: modellistica per serie temporali

Prophet è una libreria open source sviluppata da Meta (ex Facebook), progettata per la previsione di serie temporali. È stata creata per essere utilizzabile anche da chi non ha

competenze avanzate in statistica, pur offrendo un approccio solido e flessibile alla modellizzazione di dati nel tempo.

L’obiettivo principale di Prophet è quello di rendere semplice e affidabile la previsione di fenomeni che si sviluppano nel tempo, come l’andamento delle vendite, l’affluenza degli utenti o la variazione di valori economici. Funziona bene in presenza di dati con una forte componente stagionale (giornaliera, settimanale o annuale) e con tendenze non lineari, anche in presenza di dati mancanti o anomalie.

Tra le caratteristiche principali di Prophet si possono evidenziare:

- **Modello additivo**: Prophet combina tendenza, stagionalità e festività in modo additivo, cioè sommando i vari contributi, il che rende il modello interpretabile e personalizzabile.

- **Gestione automatica di outlier e cambi di trend**: la libreria è in grado di rilevare automaticamente cambiamenti significativi nell’andamento dei dati.

- **Facilità d’uso**: basta fornire un file con due colonne (`ds` per la data e `y` per il valore osservato), e il modello è già pronto per essere addestrato.

In Apollo, Prophet è stato utilizzato per fare delle previsioni riguardanti l’andamento dell’epidemia COVID-19, tenendo anche conto dei periodi festivi o delle stagionalità. Questo ha permesso di ottenere previsioni attendibili in modo rapido e con poche righe di codice, pur mantenendo una buona accuratezza, e grazie alla sua combinazione di semplicità ed efficacia, si è dimostrato uno strumento ideale per applicare la modellazione delle serie temporali anche in un contesto didattico o sperimentale.

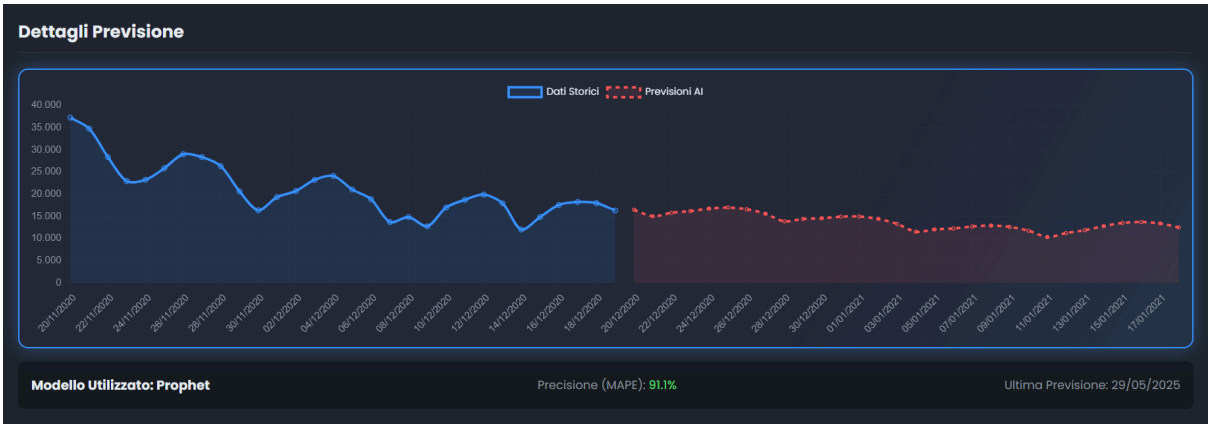


immagine relativa alle previsioni AI rispetto ai dati statistici reali

4.3 JavaScript per l'interattività (Chart.js, Three.js)

JavaScript è il linguaggio di riferimento per rendere le pagine web dinamiche e interattive. A differenza di HTML e CSS, che si occupano della struttura e dello stile, JavaScript permette di reagire agli input dell’utente, manipolare gli elementi della pagina in tempo reale e visualizzare dati in modo avanzato.

Nel contesto di Apollo, JavaScript è stato utilizzato per migliorare l’esperienza utente e rappresentare i dati in maniera chiara e coinvolgente, attraverso due librerie specifiche:

- **Chart.js**: è una libreria leggera e semplice da usare per creare grafici animati in HTML5. Supporta diversi tipi di visualizzazione, come grafici a linee, a barre, a torta e altro ancora. In questo progetto è stata impiegata per [es. “visualizzare l’andamento dei dati previsti da Prophet in modo comprensibile e interattivo”], facilitando l’interpretazione visiva delle serie temporali e delle tendenze.

- **Three.js**: è una libreria potente per la creazione di grafica 3D nel browser utilizzando WebGL. Permette di costruire scene tridimensionali, modelli interattivi e visualizzazioni complesse, mantenendo buone prestazioni. In questo progetto è stata utilizzata per la creazione del globo 3D nella quale abbiamo inserito la visualizzazione dei casi COVID-19

L’integrazione di queste tecnologie ha permesso di trasformare semplici dati numerici in rappresentazioni visive intuitive, rendendo il progetto non solo più funzionale, ma anche più comunicativo e accessibile.

4.4 HTML, CSS e Jinja2 per l'interfaccia utente

L’interfaccia utente è una componente fondamentale di qualsiasi applicazione web, in quanto rappresenta il punto di contatto diretto tra l’utente e il sistema. Per costruire una UI chiara, accessibile e funzionale, sono stati utilizzati HTML, CSS e il motore di template Jinja2, in combinazione con Flask.

- **HTML (HyperText Markup Language)** è il linguaggio standard per definire la struttura dei contenuti web. Ogni pagina del progetto è stata costruita usando elementi HTML per organizzare testi, pulsanti, grafici e altri componenti in modo semantico e ordinato.

- **CSS (Cascading Style Sheets)** è stato utilizzato per definire l’aspetto grafico dell’interfaccia, come colori, font, layout e animazioni. Grazie al CSS è stato possibile rendere il sito non solo esteticamente gradevole, ma anche responsive, cioè adattabile a diverse dimensioni di schermo.

- **Jinja2** è un motore di template integrato in Flask, che permette di generare pagine HTML dinamiche. Grazie a Jinja2 è stato possibile inserire contenuti e dati provenienti dal backend direttamente all’interno del codice HTML, usando strutture di controllo come cicli e condizioni. Questo ha permesso di visualizzare risultati aggiornati in tempo reale, come tabelle, previsioni o grafici generati dinamicamente.

L’uso combinato di questi strumenti ha reso possibile la creazione di un’interfaccia utente intuitiva, funzionale e integrata con il backend Python, offrendo così una vera esperienza full-stack.

4.5 API RESTful: Comunicazione decoupled

Le API RESTful (Representational State Transfer) rappresentano uno standard ampiamente utilizzato per la comunicazione tra sistemi software. In un'architettura moderna, le API consentono di separare la logica del backend (server) dalla presentazione del frontend (interfaccia utente), favorendo una struttura **decoupled** — ovvero indipendente e modulare.

In Apollo le API RESTful sono state impiegate per gestire lo scambio di dati tra il client (ad esempio, una pagina web o un’applicazione JavaScript) e il server realizzato in Flask. Le principali caratteristiche di questo approccio includono:

- **Comunicazione tramite HTTP**: le API utilizzano i metodi HTTP standard (GET, POST, PUT, DELETE) per inviare richieste e ricevere risposte in formato JSON, un formato leggero e facilmente interpretabile.

- **Separazione dei ruoli**: grazie all’architettura REST, l’interfaccia utente può evolversi indipendentemente dalla logica di business o dal database, favorendo la manutenzione, l’estensibilità e la riusabilità del codice.

- **Interoperabilità**: essendo basate su protocolli standard, le API RESTful possono essere utilizzate da qualsiasi client, anche sviluppato con tecnologie diverse, rendendo il sistema più aperto e flessibile.

Nel progetto, le API sono state utilizzate per richiedere i dati storici, le previsioni e altre informazioni, come l'accuratezza del modello, garantendo un flusso di dati efficiente e ben strutturato tra frontend e backend.

Questo approccio ha contribuito a rendere l’intera applicazione più modulare, scalabile e facilmente integrabile con altri sistemi.

4.6 GeoJSON per i dati geospaziali

GeoJSON è un formato basato su JSON utilizzato per rappresentare dati geospaziali, come punti, linee, poligoni e altre geometrie. È uno standard molto diffuso nel web mapping perché semplice da leggere, compatibile con molte librerie JavaScript e facilmente integrabile in applicazioni web.

Nel progetto Apollo, GeoJSON è stato impiegato per rappresentare informazioni geografiche in modo strutturato, come ad esempio la posizione dei casi reali e quelli previsti dall’IA. Ogni oggetto GeoJSON contiene coordinate geografiche (latitudine e longitudine) e, opzionalmente, attributi associati, come nomi di luoghi, valori misurati o categorie.

Le principali caratteristiche del formato GeoJSON includono:

- **Semplicità**: essendo basato su JSON, è facilmente leggibile e gestibile da qualsiasi linguaggio moderno, incluso JavaScript e Python.

- **Struttura standardizzata**: permette di rappresentare sia singole entità geografiche (Feature) che collezioni di entità (FeatureCollection), rendendo il formato adatto anche per dataset complessi.

- **Integrazione con librerie di mappatura**: GeoJSON può essere usato direttamente con librerie come Leaflet.js o Mapbox per visualizzare dati geospaziali su mappe interattive nel browser.

Nel progetto Apollo, l’uso di GeoJSON ha permesso di arricchire l’interfaccia con una componente geografica interattiva e informativa, migliorando la comprensione dei dati attraverso la loro visualizzazione spaziale.

5. **Funzionalità Principali di Apollo**

Il progetto **Apollo** nasce con l’obiettivo di unire analisi dati e visualizzazione interattiva in un’unica piattaforma, semplice da usare ma ricca di contenuti. Le sue funzionalità principali spaziano dalla previsione statistica di fenomeni reali, come l’andamento della pandemia da COVID-19, alla rappresentazione visiva su un globo 3D interattivo dei dati geospaziali.

Grazie all’integrazione di strumenti di data science, librerie di visualizzazione grafica e dati reali, Apollo permette agli utenti di esplorare informazioni complesse in modo intuitivo e coinvolgente. Nei punti seguenti verranno analizzate in dettaglio le funzionalità più significative del progetto.

5.1 Caricamento e visualizzazione dati storici

Una delle funzionalità fondamentali di Apollo è la capacità di caricare, elaborare e visualizzare dati storici relativi all’evoluzione della pandemia da COVID-19 in Italia. Il progetto utilizza come fonte i dataset ufficiali della **Protezione Civile**, che vengono forniti in formato `.csv` e aggiornati periodicamente.

I dati includono informazioni giornaliere come il numero di nuovi positivi, guariti, decessi e tamponi effettuati. Attraverso uno script Python dedicato, Apollo legge questi file, li elabora e li converte in formati adatti all’analisi e alla visualizzazione.

Una volta caricati, i dati vengono rappresentati graficamente per permettere all’utente di esplorarne l’andamento nel tempo. Questa visualizzazione iniziale è fondamentale non solo per comprendere l’evoluzione storica del fenomeno, ma anche per fornire una base solida su cui costruire eventuali previsioni statistiche, trattate nei punti successivi.

5.2 Addestramento e visualizzazione previsioni

Una delle componenti più innovative di Apollo è la possibilità di generare **previsioni sull’andamento futuro** della pandemia tramite l’uso della libreria **Prophet**, sviluppata da Facebook per l’analisi di serie temporali.

Dopo aver caricato i dati storici, il sistema li utilizza per addestrare un modello predittivo. Prophet è in grado di identificare automaticamente **trend**, **stagionalità** e **cambiamenti nei dati**, anche in presenza di dati mancanti o rumore. Questo lo rende particolarmente adatto all’analisi di fenomeni complessi e variabili come quello dell’epidemia da COVID-19.

Il risultato dell’addestramento viene poi visualizzato tramite un grafico che mostra:

- la curva storica dei dati reali,
- la proiezione futura del fenomeno (ad esempio, nuovi contagi previsti),
- un intervallo di confidenza che indica il margine di incertezza della previsione.

Questa rappresentazione rende immediatamente comprensibile l’andamento atteso e aiuta a riflettere su possibili scenari futuri. La combinazione di calcolo statistico e visualizzazione grafica rende lo strumento utile sia in ambito educativo che per una consultazione pubblica dei dati.

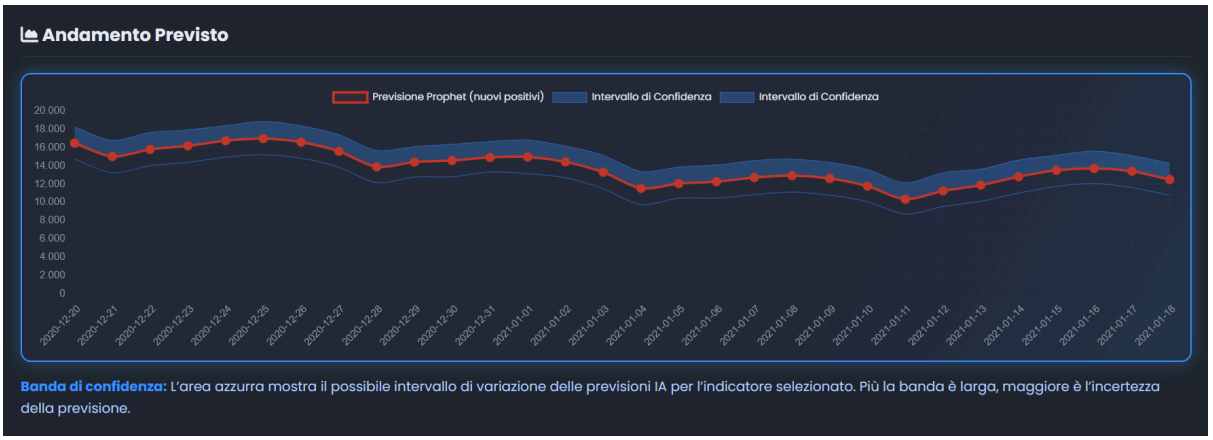


grafico relativo alle predizioni grazie a prophet con il un margine di confidenzialità

5.3 Analisi delle componenti del modello

Il modello progettuale alla base del progetto si compone di diverse componenti fondamentali, ognuna delle quali svolge un ruolo chiave nell’architettura generale dell’applicazione. La suddivisione modulare consente una maggiore manutenibilità, scalabilità e facilità di sviluppo collaborativo. Le principali componenti del modello sono le seguenti:

5.3.1 Gestione delle rotte e delle API

Nel progetto, la gestione delle rotte è affidata a Flask, un framework leggero e flessibile per lo sviluppo di applicazioni web in Python. Ogni rotta (@app.route) rappresenta un endpoint API che consente al frontend o ad altri client di interagire con il backend. Ad esempio, la rotta /api/stats/model restituisce statistiche sul modello di previsione, mentre /api/model/mape fornisce la MAPE (Mean Absolute Percentage Error) per specifici indicatori e paesi. L’uso di Flask permette di gestire facilmente le richieste HTTP, validare i parametri di input e restituire risposte in formato JSON, facilitando l’integrazione con applicazioni web o mobile.

5.3.2 Interfaccia con il database

Anche se dal codice mostrato non si evince esplicitamente la presenza di un database, solitamente un’applicazione di questo tipo utilizza un database relazionale (come SQLite, MySQL o PostgreSQL) o non relazionale (come MongoDB) per la memorizzazione dei dati. I moduli dedicati al database si occupano di gestire la connessione, eseguire query e mantenere la coerenza dei dati. Questa separazione consente di centralizzare la logica di accesso ai dati, migliorando la sicurezza e facilitando eventuali modifiche future al tipo di database utilizzato.

5.3.3 Logica di business

La logica di business è implementata tramite funzioni e classi che eseguono le operazioni principali del progetto, come l’elaborazione dei dati, il calcolo delle statistiche o la generazione delle previsioni tramite il modello Prophet. Ad esempio, la funzione get_model_stats() richiama metodi del modello di machine learning per ottenere informazioni aggiornate sulle sue performance. Mantenere la logica di business separata dalle altre componenti favorisce la chiarezza del codice e la possibilità di effettuare test automatici.

5.3.4 Gestione della sicurezza

La sicurezza viene garantita attraverso la validazione degli input ricevuti dagli endpoint API, come si vede nella funzione api_mape(), che verifica che i parametri siano tra quelli consentiti e restituisce un error 400 in caso di valori non validi. Inoltre, sono implementati controlli per gestire eventuali errori o accessi non autorizzati, proteggendo così il sistema da possibili attacchi o malfunzionamenti. In un progetto reale, si possono aggiungere ulteriori livelli di sicurezza, come l’autenticazione degli utenti, la gestione delle sessioni e la protezione contro attacchi comuni (ad esempio, SQL injection o Cross-Site Scripting).

5.3.5 Componenti ausiliarie

Infine, il progetto può includere componenti ausiliarie come moduli di logging (per tracciare errori e attività del sistema), utility per la gestione dei file o script di automazione per la manutenzione. Nel codice mostrato, ad esempio, viene utilizzato un logger per registrare eventuali errori durante il recupero delle statistiche del modello. Questi strumenti sono fondamentali per monitorare lo stato dell’applicazione, facilitare la risoluzione dei problemi e garantire la stabilità nel tempo.

Questa struttura modulare garantisce che ciascuna componente possa essere sviluppata, testata e aggiornata in modo indipendente, contribuendo così all’affidabilità e alla robustezza complessiva del progetto.

5.4 Valutazione delle performance del modello

La valutazione delle performance del modello rappresenta una fase cruciale nello sviluppo di sistemi di previsione, in quanto consente di misurare l’accuratezza delle stime prodotte e di identificare eventuali margini di miglioramento. Nel progetto, la performance del modello viene monitorata attraverso specifici indicatori statistici, tra cui la MAPE (Mean Absolute Percentage Error), che quantifica l’errore percentuale medio tra i valori previsti e quelli reali.

Per ogni previsione effettuata, il sistema confronta i dati generati dal modello con i dati effettivamente osservati, calcolando così l’errore commesso. La MAPE, in particolare, è stata scelta per la sua interpretabilità e per la facilità con cui permette di confrontare le performance su diversi indicatori e paesi. Un valore di MAPE basso indica una buona capacità predittiva del modello, mentre valori elevati suggeriscono la necessità di ricalibrare o migliorare l’algoritmo.

Oltre alla MAPE, il sistema può restituire ulteriori statistiche relative al modello, come la varianza residua, l’accuratezza complessiva e altri indicatori di bontà del fit. Questi dati sono accessibili tramite apposite API, che permettono di monitorare in tempo reale le performance e di adattare la strategia di previsione in base ai risultati ottenuti.

La valutazione continua delle performance è fondamentale per garantire l’affidabilità del sistema, soprattutto in contesti dinamici come la previsione dell’andamento di una pandemia, dove i dati possono variare rapidamente e il modello deve essere costantemente aggiornato e validato.

visualizzazione della precisione del modello AI grazie a MAPE

5.5 Esplorazione dati su mappamondo 3D interattivo

La visualizzazione dei dati su un globo 3D si basa su un sistema integrato tra frontend e backend che permette un’interazione dinamica e immediata con le informazioni geografiche e temporali. Quando l’utente seleziona un indicatore o una fascia temporale di interesse, il sistema invia una richiesta al backend, il quale recupera e prepara i dati aggiornati per le diverse nazioni.

Dal lato **frontend**, la rappresentazione dei dati è curata nei minimi dettagli per offrire un’esperienza visiva intuitiva e coinvolgente:

- Il globo viene mostrato in 3D, con possibilità di navigazione libera (rotazione, zoom) attorno alla Terra.
- Ogni paese è rappresentato tramite elementi grafici come marker, colonne o variazioni di colore, a seconda del valore dell’indicatore selezionato.
- Sono presenti tooltip e finestre di dettaglio che mostrano informazioni aggiuntive senza appesantire la vista principale.
- L’interfaccia permette di filtrare i dati e di osservare trend temporali tramite animazioni fluide.

Il **backend** svolge un ruolo fondamentale nella gestione e nella preparazione dei dati:

- Riceve le richieste provenienti dal frontend e accede al database per estrarre i dati necessari.
- Elabora e organizza i dati per essere restituiti in modo efficiente e rapido.
- Supporta il sistema con API ottimizzate per garantire risposte in tempo reale.

Per mantenere prestazioni elevate anche con dataset di grandi dimensioni, il modello sfrutta alcune strategie tecniche fondamentali:

- Caricamento asincrono dei dati, che evita blocchi o rallentamenti nell’interfaccia utente.
- Ottimizzazione del rendering 3D, con gestione intelligente delle geometrie per ridurre il carico computazionale, mantenendo comunque una qualità visiva elevata.

Questa soluzione avanzata trasforma la consultazione di semplici tabelle numeriche in un’esperienza immersiva, stimolando la curiosità e facilitando l’analisi comparativa tra aree geografiche e periodi storici diversi.

6. Sviluppo, Testing e Documentazione

Per la realizzazione di Apollo: lo sviluppo del software, le strategie di testing adottate e la produzione della documentazione tecnica. L’obiettivo è illustrare il percorso seguito dalla progettazione iniziale fino alla messa in produzione, evidenziando le scelte metodologiche, gli strumenti utilizzati e le pratiche adottate per garantire la qualità, la robustezza e la manutenibilità dell’applicazione. Verranno inoltre analizzate le modalità di verifica delle funzionalità implementate e l’importanza di una documentazione chiara e accessibile per facilitare l’utilizzo e l’eventuale evoluzione futura del sistema.

6.1 Metodologia di sviluppo

Lo sviluppo di Apollo si è basato su un approccio strutturato e iterativo, volto a garantire qualità, flessibilità e facilità di manutenzione del prodotto finale. Il lavoro è stato suddiviso in cicli incrementali, ognuno focalizzato su specifiche funzionalità o componenti, permettendo così di monitorare costantemente i progressi e intervenire tempestivamente in caso di criticità.

Dal punto di vista tecnologico, Apollo è stato realizzato utilizzando Python per il backend, sfruttando il framework Flask per la gestione delle API e delle rotte server-side. Per la parte di analisi e previsione dei dati, è stato integrato il modello Prophet, particolarmente adatto alla modellazione di serie temporali. La gestione dei dati e delle statistiche avviene tramite moduli dedicati, che si interfacciano con il backend per fornire informazioni aggiornate e accurate.

Il frontend dell’applicazione è stato sviluppato utilizzando tecnologie web moderne, tra cui JavaScript e la libreria Three.js, che ha permesso di realizzare la visualizzazione interattiva dei dati su un globo 3D. L’integrazione tra frontend e backend avviene tramite chiamate API RESTful, garantendo così una comunicazione efficiente e scalabile tra le diverse componenti del sistema.

Durante tutto il ciclo di sviluppo, sono stati utilizzati strumenti di versionamento del codice (come Git) e ambienti di sviluppo collaborativi, che hanno facilitato la condivisione del lavoro e la gestione delle modifiche. La documentazione delle scelte progettuali e delle funzionalità implementate è stata curata costantemente, così da rendere il progetto facilmente comprensibile e mantenibile anche in futuro.

In sintesi, la metodologia adottata ha permesso di integrare in modo efficace diverse tecnologie e competenze, portando alla realizzazione di un sistema robusto, moderno e facilmente espandibile.

6.2 Testing delle API Backend con Pytest

Per garantire l’affidabilità e la correttezza delle API backend di *Apollo*, è stato adottato **Pytest**, uno dei framework di testing più diffusi e potenti nell’ecosistema Python. Pytest consente di scrivere test in modo semplice e leggibile, facilitando l’individuazione di eventuali bug o regressioni durante lo sviluppo.

I test sono organizzati in file separati (ad esempio `test_api.py`) all’interno di una cartella dedicata chiamata `tests/`. Ogni funzione di test verifica il comportamento di uno o più endpoint API, simulando richieste HTTP e controllando che le risposte siano corrette sia dal punto di vista del contenuto che del codice di stato restituito.

Esempio di test per l’endpoint `/api/stats/model`:

```
import pytest
from app import app

@pytest.fixture
def client():
    with app.test_client() as client:
        yield client

def test_get_model_stats(client):
    response = client.get('/api/stats/model')
    assert response.status_code in [200, 404]
    data = response.get_json()
    assert 'success' in data
```

Per eseguire tutti i test, è sufficiente lanciare il comando dalla root del progetto:

```
pytest
```

Pytest individuerà automaticamente tutti i file che iniziano con `test_` ed eseguirà le relative funzioni di test, mostrando un riepilogo dei risultati direttamente in console. In caso di errori o fallimenti, verranno riportati dettagli utili per la diagnosi e la correzione dei problemi.

L’impiego di Pytest ha permesso di mantenere un alto livello di affidabilità nel backend di *Apollo*, facilitando il refactoring del codice e l’aggiunta di nuove funzionalità senza introdurre errori imprevisti.

6.3 Documentazione tecnica del progetto

La documentazione tecnica di **Apollo** è stata realizzata con l’obiettivo di rendere il progetto facilmente comprensibile, utilizzabile e manutenibile, sia dagli sviluppatori che da eventuali collaboratori futuri. Essa copre tutti gli aspetti fondamentali del sistema: dalle tecnologie adottate, alla struttura dei file, fino al dettaglio delle API e delle strategie di testing.

6.3.1 Tecnologie e architettura

Apollo si basa su una moderna architettura **web full-stack**:

- **Frontend**: sviluppato con HTML, CSS e JavaScript (ES6+), con l'integrazione di:
  - `Chart.js` per la visualizzazione di grafici;
  - `Three.js` per la rappresentazione interattiva del globo 3D.
- **Backend**: realizzato in Python, utilizza il framework **Flask** per esporre API RESTful che permettono la comunicazione tra client e server.
- **Modello di previsione**: viene utilizzato **Prophet** (sviluppato da Meta/Facebook), particolarmente adatto all’analisi e previsione di serie temporali.
- **Gestione dei dati**: i dati sono caricati da file **CSV ufficiali della Protezione Civile** e gestiti con la libreria **Pandas**.

6.3.2 Struttura dei file

La documentazione descrive la struttura dei principali file del progetto:

- `server/app.py`: punto di ingresso dell'applicazione Flask; contiene la definizione delle rotte e delle API.
- `server/models/prophet_model.py`: definisce la classe `ProphetModel`, responsabile del training del modello, delle previsioni e del calcolo delle bande di confidenza.
- `server/data_utils.py`: contiene funzioni di supporto per il pre-processing e la manipolazione dei dati.
- `requirements.txt`: elenca tutte le dipendenze Python necessarie per eseguire il progetto.

6.3.3 API e flusso dei dati

Sono stati documentati in dettaglio i principali **endpoint API**:

- `/api/data/historical`: restituisce i dati storici.
- `/api/data/forecast`: fornisce le previsioni future.
- `/api/model/mape`: calcola l’accuratezza delle previsioni (Mean Absolute Percentage Error).
- `/api/data/latest`: restituisce i dati più recenti disponibili.
- `/api/data/globe`: fornisce i dati aggregati da visualizzare sul globo 3D.

Per ciascun endpoint vengono specificati:

- i parametri richiesti;
- il formato della risposta JSON;
- i possibili codici di errore restituiti.

6.3.4 Logging, debug e testing

La documentazione illustra anche le strategie adottate per:

- **Logging**: per registrare attività ed errori, facilitando la manutenzione e la risoluzione dei problemi.
- **Testing automatico**: implementato con **Pytest**, include esempi pratici di test delle API e istruzioni per l'esecuzione dei test tramite il comando: `pytest`

6.3.5 Strumenti di supporto

Infine, vengono fornite indicazioni utili per:

- l’installazione delle dipendenze;
- la configurazione dell’ambiente di sviluppo;
- l’avvio dell’applicazione;
- la consultazione di ulteriori risorse, come il file `README.md` e i commenti nel codice.

Questa documentazione rappresenta un **punto di riferimento fondamentale** per chiunque voglia comprendere, utilizzare o contribuire allo sviluppo di *Apollo*, garantendo la trasparenza e la continuità del progetto nel tempo.

7. Risultati e Analisi

7.1 Capacità Predittive e di Visualizzazione

Il progetto **Apollo** integra in modo efficace un modello predittivo statistico con strumenti di visualizzazione avanzati. Il modello **Prophet** permette di elaborare previsioni attendibili riguardo l’evoluzione di indicatori epidemiologici fondamentali, tra cui nuovi casi, decessi e guarigioni. Tali previsioni vengono presentate all’utente tramite un’interfaccia interattiva che facilita l’analisi anche da parte di utenti non esperti.

L’interfaccia offre una **doppia modalità di visualizzazione**:

- **Grafici temporali**: mostrano l’andamento storico e le previsioni, permettendo un confronto diretto tra valori previsti e reali (realizzati con Chart.js).

- **Globo 3D interattivo**: visualizza la distribuzione geografica dei dati e delle previsioni, permettendo una panoramica globale intuitiva (basato su Three.js).

Entrambi gli strumenti offrono filtri personalizzabili per indicatore, periodo o area geografica, e rendono possibile identificare rapidamente **pattern, anomalie e tendenze** difficili da notare in una semplice tabella numerica. L’interattività contribuisce a trasformare la consultazione dei dati in un’esperienza dinamica ed esplorativa.

7.2 Interpretazione delle Metriche di Errore (MAE, RMSE, MAPE)

Per valutare l’affidabilità delle previsioni, Apollo impiega **tre metriche di errore**, ognuna utile a cogliere aspetti diversi della performance del modello:

- **MAE (Mean Absolute Error)**: misura l’errore medio assoluto tra valori reali e previsti. È facile da interpretare e indica lo scostamento medio.

- **RMSE (Root Mean Squared Error)**: penalizza maggiormente gli errori più grandi, risultando più severa in presenza di anomalie.

- **MAPE (Mean Absolute Percentage Error)**: esprime l’errore in termini percentuali, rendendo possibile il confronto tra variabili di scala diversa.

Queste metriche vengono calcolate a vari livelli:

- Globalmente, per un’analisi complessiva.
- Per ogni indicatore, al fine di capire se il modello è più efficace su alcuni dati rispetto ad altri.
- Per singolo paese, utile nei confronti internazionali.

I risultati ottenuti dimostrano una **buona accuratezza predittiva**, anche se in presenza di serie temporali molto instabili o di dati storici ridotti, si possono osservare scostamenti più significativi. In tali casi, i risultati delle metriche aiutano a identificare possibili ottimizzazioni future o a motivare un aggiornamento dei dati.

L’utilizzo combinato di MAE, RMSE e MAPE offre quindi una visione approfondita sull’efficacia del sistema, rendendo possibile **migliorare il modello nel tempo** e aiutando l’utente finale a **valutare criticamente l’affidabilità delle previsioni**.

8. Conclusioni e Sviluppi Futuri

8.1 Esito finale del Progetto e Obiettivi Raggiunti

Il progetto **Apollo** ha raggiunto con successo gli obiettivi prefissati, dimostrando la validità dell’idea iniziale e l’efficacia della sua realizzazione tecnica. Fin dalle fasi iniziali, l’intento era quello di creare un sistema che unisse l’elaborazione di previsioni epidemiologiche tramite l’intelligenza artificiale con una modalità di visualizzazione interattiva, moderna e intuitiva. Questo traguardo è stato pienamente centrato.

Dal punto di vista funzionale, Apollo consente all’utente di accedere a previsioni attendibili sui dati pandemici, sfruttando il modello Prophet per analizzare e proiettare l’andamento di indicatori cruciali come contagi, decessi e guariti. L'interfaccia permette una fruizione semplice ma approfondita, sia attraverso grafici temporali dettagliati, sia tramite un globo 3D esplorabile, che facilita la comprensione dell’impatto geografico della pandemia

8.2 Punti di Forza del progetto Apollo

Il progetto **Apollo** presenta numerosi punti di forza che ne hanno determinato l’efficacia e la completezza, sia dal punto di vista tecnico che didattico.

Dal punto di vista tecnico, si evidenziano:

- **Un backend solido**, sviluppato in Python con Flask, capace di gestire richieste API, elaborare dati epidemiologici reali e restituire previsioni accurate in modo veloce ed efficiente.

- **Un frontend interattivo e responsive**, costruito con HTML, CSS e JavaScript, che permette una navigazione fluida e intuitiva. Grazie all’integrazione di librerie come Chart.js e Three.js, l’interfaccia risulta chiara anche per utenti non esperti.

- **Testing automatizzato con Pytest**, fondamentale per garantire l’affidabilità e la stabilità del sistema. I test permettono di identificare subito eventuali errori, semplificando la manutenzione e lo sviluppo di nuove funzionalità.

- **Documentazione tecnica completa**, redatta per spiegare in modo chiaro la struttura del progetto, le dipendenze, gli endpoint API e le modalità d’uso. Questo rende Apollo facilmente comprensibile e ampliabile anche da altri sviluppatori.

Dal punto di vista formativo e personale, Apollo è stato un progetto che ha permesso di mettere in pratica competenze trasversali, offrendo un’esperienza multidisciplinare:

- **Programmazione full-stack**, unendo la gestione del backend (Python, Flask) con lo sviluppo del frontend (JavaScript, Three.js).

- **Data science e machine learning**, grazie all’uso del modello Prophet e alla manipolazione dei dati con Pandas.

- **Visualizzazione dati interattiva**, con l’obiettivo di rendere l’analisi delle previsioni accessibile e coinvolgente.

Questi elementi, uniti alla forte motivazione personale nella realizzazione del progetto, rappresentano i principali punti di forza di Apollo. Il risultato finale è un sistema completo, chiaro e facilmente estendibile, che coniuga tecnologia, usabilità e utilità.

8.3 Prospettive future e possibili miglioramenti

Uno degli aspetti più interessanti del progetto Apollo è la sua struttura modulare e scalabile, che ne consente un’evoluzione continua anche dopo il completamento della versione attuale. Sono numerosi, infatti, gli sviluppi futuri immaginati per arricchirne le funzionalità e ampliare il campo di applicazione.

Tra le principali **prospettive di miglioramento**, si segnalano:

- **Estensione dei dati a livello regionale e internazionale**: attualmente Apollo lavora principalmente su dati aggregati nazionali. L’integrazione di fonti regionali o provenienti da altri paesi permetterebbe una visione più dettagliata e comparativa, utile ad esempio per analizzare focolai locali o confrontare l’efficacia delle misure adottate in diverse aree.

- **Evoluzione del modello predittivo**: sebbene Prophet offra già buone prestazioni, è possibile esplorare modelli più sofisticati come reti neurali (LSTM, GRU) o approcci ensemble che combinano più modelli predittivi. Ciò migliorerebbe la precisione delle stime, soprattutto in contesti più complessi o instabili.

- **Dashboard personalizzabili**: implementare un’interfaccia adattabile, dove l’utente può scegliere quali grafici visualizzare, impostare alert su soglie personalizzate o salvare viste preferite, aumenterebbe notevolmente l’utilità per utenti esperti, analisti o decisori.

- **Integrazione di pratiche DevOps**: attraverso sistemi di integrazione continua (CI/CD) e automazione dei test, si potrebbe migliorare il ciclo di sviluppo e rilascio. L’introduzione di strumenti come GitHub Actions o Jenkins renderebbe il progetto più robusto e professionale.

- **Funzionalità avanzate di esportazione**: l’aggiunta di strumenti per esportare dati e grafici in vari formati (PDF, CSV, PNG) renderebbe Apollo utile anche per chi ha necessità di documentare le analisi, preparare report o condurre ricerche.

Queste migliorie, pur non essendo attualmente implementate, rappresentano una **naturale evoluzione** del progetto, in grado di trasformarlo da semplice applicazione didattica a strumento avanzato per l’analisi predittiva e la visualizzazione interattiva di dati epidemiologici e non solo. Il potenziale di Apollo, infatti, può essere esteso anche ad altri ambiti, come l’economia, l’ambiente o i trasporti, semplicemente modificando la fonte dei dati e il tipo di indicatori da analizzare.

9. Bibliografia e Sitografia Essenziale

- Documentazione ufficiale di **Flask**: https://flask.palletsprojects.com/
- Documentazione ufficiale di **Prophet**: https://facebook.github.io/prophet/
- Documentazione ufficiale di **Pandas**: https://pandas.pydata.org/pandas-docs/stable/
- Documentazione ufficiale di **Scikit-learn**: https://scikit-learn.org/stable/documentation.html
- Documentazione ufficiale di **Chart.js**: https://www.chartjs.org/docs/latest/
- Documentazione ufficiale di **Three.js**: https://threejs.org/docs/
- Dati COVID-19 Protezione Civile: Repository GitHub della Protezione Civile (https://github.com/pcm-dpc/COVID-19)