



NoSchoolQL

Corso di Basi Dati II
AS 2024-2025



TIMELINE PRESENTAZIONE

Una panoramica organizzata di NoSchoolQL: una piattaforma per la creazione e l'amministrazione di dataset scolastici sintetici, ottimizzata per database NoSQL.

Introduzione e Motivazioni

Use Case & Requisiti Utente

Definizione del Mini-world e Dataset

Pipeline ETL in Python

Modello di Calcolo dei Voti

Architettura Full-Stack

Risultati & Conclusioni

TIMELINE PRESENTAZIONE

In questa sezione si contestualizza NoSchoolQL all'interno del corso di Basi di Dati II, evidenziando il bisogno di sperimentare pipeline ETL, modellazione NoSQL e architetture full-stack su dati scolastici sintetici.

Introduzione e Motivazioni



Use Case & Requisiti Utente



Definizione del Mini-world e Dataset



Pipeline ETL in Python



Modello di Calcolo dei Voti



Architettura Full-Stack

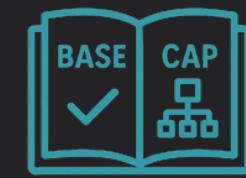


Risultati & Conclusioni

INTRODUZIONE & MOTIVAZIONI

NoSchoolQL nasce in Basi di Dati II per sperimentare modelli NoSQL, pipeline ETL Python e architettura full-stack Node.js/React sui dati MIM.

Contesto Didattico



Applicazione pratica di BASE, CAP e modellazione documentale.

Motivazioni EdTech



Gestione real-time di anagrafica, voti e statistiche.

Obiettivi di Apprendimento



Implementare pipeline ETL e SPA React con API REST.

MOTIVAZIONI DEL PROGETTO

Nel contesto EdTech, un registro elettronico moderno deve saper gestire volumi elevati di dati disparati, garantire tempi di risposta rapidi e rimanere disponibile anche in caso di interruzioni di rete. NoSchoolQL è nato per rispondere a queste esigenze con un approccio modulare e scalabile.



Dati Eterogenei

Supporto simultaneo ad anagrafica, voti e statistiche aggregate, senza schema fisso.



Interfaccia Reattiva

SPA React che assicura un'esperienza fluida per studenti e docenti, con caricamenti asincroni.



Alta Disponibilità

Backend stateless e replica set MongoDB per tolleranza alle partizioni e fail-over trasparente.



Estensibilità Modulare

Pipeline ETL Python e API REST disaccoppiate, pronte per future integrazioni o microservizi.

TIMELINE PRESENTAZIONE

In questa sezione definiamo gli attori principali, gli use case e i requisiti funzionali e non funzionali che guidano la progettazione e lo sviluppo di NoSchoolQL.



Introduzione e Motivazioni



Use Case & Requisiti Utente



Definizione del Mini-world e Dataset



Pipeline ETL in Python



Modello di Calcolo dei Voti



Architettura Full-Stack



Risultati & Conclusioni

USE CASE & REQUISITI UTENTE

L'applicazione soddisfa due profili principali – Studente e Docente – e si fonda su un insieme di requisiti funzionali e non funzionali che ne garantiscono correttezza, sicurezza e prestazioni.



Use Case Studente

Accesso con ID e visualizzazione immediata di medie per materia, distribuzione voti e storico dettagliato.



Use Case Docente

Login con ID, selezione di classe e materia e gestione completa (inserimento, modifica, cancellazione) delle valutazioni.



Requisiti Funzionali

Operazioni CRUD su Studenti, Docenti, Classi e Voti con filtri per periodo e materia, dashboard interattive.



Requisiti Non Funzionali

Tempo di risposta API < 2 sec, architettura stateless scalabile, autenticazione JWT e protezione CORS.

TIMELINE PRESENTAZIONE

In questa sezione descriviamo il “mini-world” del progetto, illustrando come vengono filtrati e normalizzati i CSV MIM originali per costruire i dataset puliti alla base della simulazione scolastica.





DEFINIZIONE DEL MINI-WORLD & DATASET

Filtraggio delle scuole

Solo gli istituti di “Scuola Secondaria di II grado” vengono estratti da Stu_Corso_Classe_Genere.csv e utilizzati come chiave per includere nei passaggi successivi soltanto i dati pertinenti.

Creazione dei dataset puliti

I file originali MIM sono normalizzati (snake_case, rimozione di duplicati e valori nulli) e salvati in dataset_puliti/, garantendo uniformità di formato.

Definizione delle entità

Dai dati puliti nascono tre concetti principali – Scuola (codice e dati anagrafici), Percorso formativo (indirizzo e anno), Aggregato studentesco (sesso, cittadinanza) – collegati implicitamente tramite il campo codicescuola.

TIMELINE PRESENTAZIONE

In questa sezione viene presentata la pipeline ETL sviluppata in Python, composta da quattro script modulari coordinati da main.py per pulire, normalizzare, simulare e validare i dati MIUR prima del caricamento in MongoDB.

Introduzione e Motivazioni

Use Case & Requisiti Utente

Definizione del Mini-world e Dataset

Pipeline ETL in Python

Modello di Calcolo dei Voti

Architettura Full-Stack

Risultati & Conclusioni

PIPELINE ETL IN PYTHON

La pipeline ETL è orchestrata da main.py, che esegue in sequenza quattro script indipendenti. Questo garantisce atomicità (interruzione al primo errore), modularità (ogni fase è autonoma e manutenibile) e riproducibilità dei risultati grazie a un seed fisso e all'uso di Faker.

Quattro fasi chiave



1. pulizia_mim.py

Normalizza CSV grezzi (snake_case, rimozione duplicati) e filtra solo le scuole di II grado, producendo tre dataset puliti.



2. calcolo_statistiche.py

Aggrega i dati puliti calcolando percentuali di maschi/femmine e italiani/stranieri, esportando statistiche_base.csv.



3. genera_dati_simulati.py

Simula classi, studenti, docenti e voti usando Faker e parametri socio-demografici, salvando i CSV definitivi.



4. analisi_dataset.py

Unisce voti e profili studenti per generare statistiche di verifica (medie per ESCS, area, cittadinanza) e ne stampa il confronto con i pattern attesi.

FASE 1: PULIZIA E NORMALIZZAZIONE

Lo script pulizia_mim.py estraе i codici delle scuole secondarie di secondo grado, normalizza nomi e colonne in snake_case, filtra gli altri CSV mantenendo solo i record pertinenti e salva i risultati uniformi in dataset_puliti/.

In modalità ridotta viene inoltre eseguito uno stratified sampling di x scuole (dove x è scelto dallo sviluppatore) per velocizzare le elaborazioni senza perdere rappresentatività.

```
# =====
# FASE 1: IDENTIFICAZIONE SCUOLE SECONDARIE DI II GRADO
# =====
"""
Prima di tutto devo identificare quali sono le scuole secondarie di II grado.
Questo mi serve per filtrare tutti gli altri dataset e lavorare solo con
le scuole di nostro interesse.

"""

# Carico il file che contiene l'ordine di scuola
df_ordini = pd.read_csv(PATH_STU_CORSO_CLASSE, dtype=str)
df_ordini = clean_string_columns(df_ordini)
df_ordini = snake_case_columns(df_ordini)

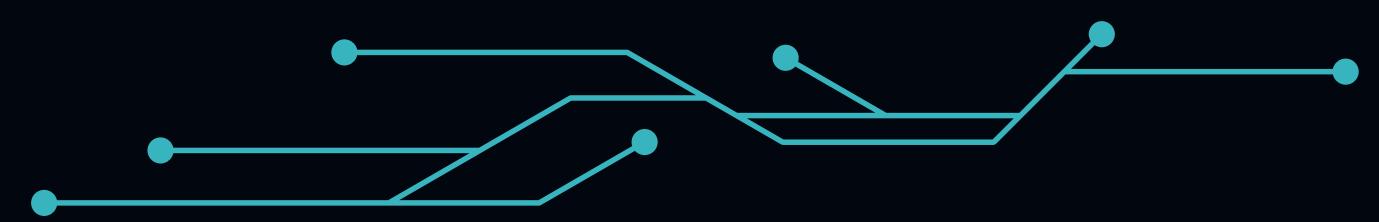
# Estraggo solo i codici delle scuole secondarie di II grado
# Questo set mi servirà per filtrare tutti gli altri dataset
scuole_secondarie = set(df_ordini[df_ordini['ordinescuola']
| | | | | == 'SCUOLA SECONDARIA II GRADO']['codicescuola'])

# =====
# FASE 2: PULIZIA ANAGRAFICA SCUOLE
# =====
"""

L'anagrafica delle scuole contiene molte informazioni non necessarie per la
simulazione. Rimuovo le colonne superflue e pulisco i dati essenziali.

"""

# Colonne da rimuovere perché non necessarie per la simulazione
drop_cols_anag = [
```



```
# =====
# FASE 1: CONVERSIONE CAMPI NUMERICI
# =====
"""
I dati MIUR a volte contengono valori non numerici o mancanti nei campi
che dovrebbero essere numerici. Li converto in modo sicuro.

"""

# Converto i campi numerici del dataset cittadinanza
for col in ['alunni', 'alunnicittadinanzaitaliana', 'alunnicittadinanzanonitaliana']:
    df_cittadinanza[col] = df_cittadinanza[col].map(to_int_safe)

# Converto i campi numerici del dataset indirizzi
for col in ['alunnimaschi', 'alunnifemmine']:
    df_indirizzi[col] = df_indirizzi[col].map(to_int_safe)

# Calcolo il totale studenti per controllo incrociato
df_indirizzi['totale'] = df_indirizzi['alunnimaschi'] + df_indirizzi['alunnifemmine']

# =====
# FASE 2: AGGREGAZIONE DATI CITTADINANZA
# =====
"""
Aggrego i dati di cittadinanza per scuola, sommando tutti gli anni di corso.
Questo mi dà il totale complessivo di studenti italiani e stranieri per scuola.

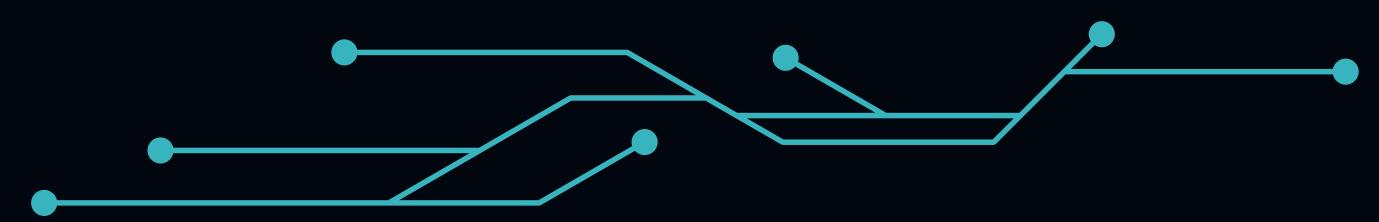
"""

agg_cittad = df_cittadinanza.groupby('codicescuola').agg({
    'alunni': 'sum', # Totale studenti
    'alunnicittadinanzaitaliana': 'sum', # Totale italiani
    'alunnicittadinanzanonitaliana': 'sum' # Totale stranieri
}).reset_index()
```

FASE 2:

CALCOLO STATISTICHE DI BASE

Lo script calcolo_statistiche.py elabora i CSV puliti per calcolare, per ciascuna scuola, le percentuali di studenti maschi, femmine, italiani e stranieri. I risultati vengono raccolti in statistiche_base.csv e forniscono i pesi di genere e cittadinanza utilizzati nelle fasi successive per garantire coerenza con i dati MIM.



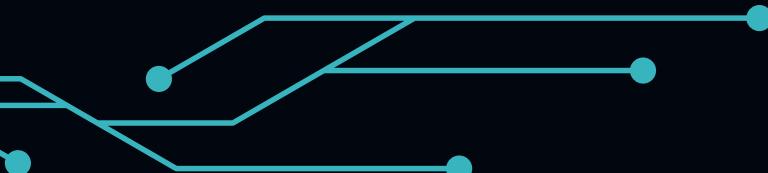
FASE 3: GENERAZIONE DEI DATI SIMULATI

Lo script `genera_dati_simulati.py` carica i CSV intermedi da `dataset_puliti/` e `statistiche_base.csv`, imposta un seed fisso per garantire riproducibilità e utilizza Faker per popolare i nomi.

In questa fase vengono generate:

- Classi e Studenti: calcolo del numero di classi per scuola, suddivisione equa degli alunni, simulazione di sesso, cittadinanza ed ESCS con quartili, esportando `classi.csv` e `studenti.csv`.
- Assegnazioni: determinazione delle materie per ciascuna classe, creazione di docenti fintizi e mappatura (file `docenti.csv` e `assegnazioni_docenti.csv`).
- Voti: applicazione del modello multicomponente per ogni studente-materia-docente, con rumoristica e check su intervallo [1-10], e salvataggio in `voti.csv`.

Tutti i CSV finali vengono scritti nella cartella `dataset_definitivi/`.



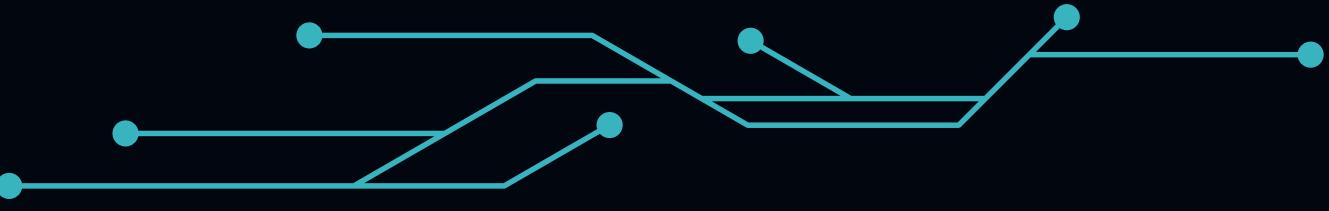
```
# -----
# FASE 5: GENERAZIONE VOTI
# -----
"""
Genero i voti degli studenti considerando molteplici fattori:
- Abilità individuale dello studente
- Difficoltà della materia
- Fattori socio-demografici (ESCS, area geografica, cittadinanza)
- Effetto classe e docente
- Tipologia di valutazione (scritto/orale/pratico)
"""

print('Generazione voti con integrazione fattori socio-demografici...')

# Configurazione temporale per le date dei voti
DATA_INIZIO = datetime.date(2023, 9, 15) # Inizio anno scolastico
DATA_FINE = datetime.date(2024, 5, 31) # Fine anno scolastico
DELTA_GIORNI = (DATA_FINE - DATA_INIZIO).days

# Preparo le strutture dati per l'accesso efficiente
assegnazioni_per_classe = defaultdict(list)
for _, r in df_assegnazioni.iterrows():
    assegnazioni_per_classe[r['id_classe']].append((r['id_docente'], r['materia']))

voti_records = []
voto_counter = 1
```



FASE 4:

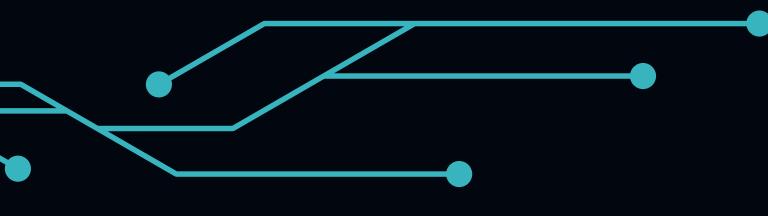
ANALISI E
VALIDAZIONE FINALE

Lo script analisi_dataset.py unisce i file voti.csv e studenti.csv sui campi id_studente, calcola le medie dei voti per cittadinanza, quartile ESCS e area geografica, e confronta i risultati con i pattern attesi dai report ufficiali, stampando a console le statistiche di verifica per garantire la coerenza del modello simulativo.

```
# =====
# STATISTICHE GENERALI SUL DATASET
# =====
"""
Produco statistiche riassuntive per dare una visione d'insieme del
dataset generato e verificare che tutti i componenti siano stati
creati correttamente.
"""

print("\n★ STATISTICHE GENERALI SUL DATASET SIMULATO:")

# Conto le entità principali
print(f"🏡 Numero scuole simulate: {df_classi['codicescuola'].nunique()}")
print(f"✍️ Numero classi: {len(df_classi)}")
print(f"👤 Numero studenti: {len(df_studenti)}")
print(f"👨‍🏫 Numero docenti: {len(df_docenti)}")
print(f"📊 Materie totali: {df_docenti['materia'].nunique()}")
print(f"📘 Assegnazioni docenti-classe: {len(df_assegnazioni)}")
print(f"คะแน Number voti registrati: {len(df_voti)}")
```



TIMELINE PRESENTAZIONE

In questa sezione viene illustrato il modello statistico utilizzato da `genera_dati_simulati.py` per calcolare i voti, attraverso la combinazione di baseline, difficoltà materia, offset classe/docente, abilità e inclinazioni dello studente, tipologia di prova, fattori socio-demografici e un termine di rumore casuale.

Introduzione e Motivazioni

Use Case & Requisiti Utente

Definizione del Mini-world e Dataset

Pipeline ETL in Python

Modello di Calcolo dei Voti

Architettura Full-Stack

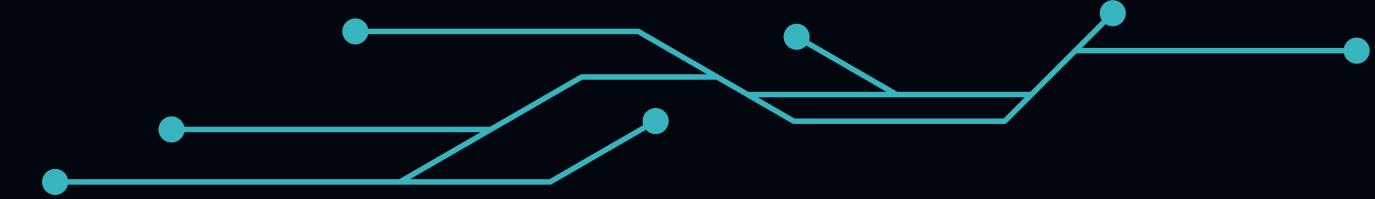
Risultati & Conclusioni

MODELLO DI CALCOLO DEI VOTI

Il voto viene calcolato in sette passaggi chiave:

1. Punto di partenza: 6.5 come media scolastica.
2. Fattore materia: si aggiunge o sottrae in base alla difficoltà (es. -1 per Matematica, +0.7 per Scienze Motorie).
3. Influenza classe/docente: un piccolo “bonus” o “malus” casuale (gaussiana $\sigma 0.4$) per riflettere lo stile d’insegnamento.
4. Influenza studente: abilità generale ($\sigma 0.6$) e inclinazione personale verso la materia ($\sigma 0.3$).
5. Tipo di prova: piccola correzione se è uno scritto, un orale o un pratico.
6. Fattore socio-demografico: media di quattro impatti (area, scuola, cittadinanza, ESCS) per simulare condizioni sociali ed economiche.
7. Rumore finale: un’ulteriore variazione casuale ($\sigma 0.7$) per rendere i voti più naturali.

Infine, il risultato viene arrotondato, mai sotto 3 (soft-floor), mai oltre 10, e gli viene assegnata una data casuale nell’anno scolastico.



```

# Componenti del voto
base = BASE_MEDIA # 6.5
diff = MATERIA_DIFFICOLTA[m_up] # Difficoltà della materia
cls_off = offset_classe_materia.get((id_classe, m_up), 0.0) # Effetto classe
stud = abilita_studente.get(id_studente, 0.0) # Abilità generale studente

# Genero o recupero la specificità studente-materia
key_spec = (id_studente, m_up)
if key_spec not in spec_studente_materia:
    # Ogni studente può essere particolarmente bravo o scarso in una materia
    spec_studente_materia[key_spec] = tronca(random.gauss(0, 0.3), -0.7, 0.7)
spec = spec_studente_materia[key_spec]

# Aggiustamento per tipologia
tip_adj = tipologia_delta(materia, tipologia)

# Impatto socio-demografico
socio_demografico = calcola_socio_demografico(id_studente)

# Rumore casuale per variabilità
noise = random.gauss(0, 0.7)

# Calcolo il voto finale sommando tutti i contributi
val = base + diff + cls_off + stud + spec + tip_adj + socio_demografico + noise

# Applico un "soft floor" a 3 per evitare troppi voti molto bassi
if val < 3 and random.random() < 0.6:
    val = 3 + random.random() * 1.0

# Limite al range valido e arrotondo
val = max(PESO_MIN_VOTO, min(PESO_MAX_VOTO, val))
return int(round(val))

```



TIMELINE PRESENTAZIONE

In questa sezione descriviamo l'architettura full-stack dell'applicazione, articolata in un frontend React per l'interfaccia utente, un'API RESTful Express per la logica applicativa e un database MongoDB in replica set per la persistenza dei dati.

Introduzione e Motivazioni

Use Case & Requisiti Utente

Definizione del Mini-world e Dataset

Pipeline ETL in Python

Modello di Calcolo dei Voti

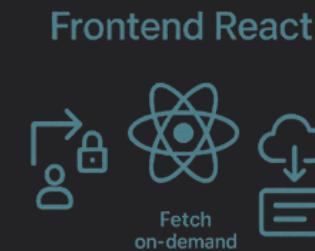
Architettura Full-Stack

Risultati & Conclusioni

ARCHITETTURA FULL-STACK

Il sistema è organizzato su tre livelli distinti: Single-Page Application React per la UI, API RESTful Node.js/Express per la logica applicativa e un database MongoDB in replica set per la persistenza.

Questa struttura garantisce chiarezza di responsabilità, scalabilità e facilità di manutenzione.



Frontend React

Routing protetto, stato in Context, fetch on-demand e placeholder per caricamenti.



Backend Node.js/Express

Middleware CORS/JSON/JWT e controller modulari che operano su collezioni MongoDB.



Database MongoDB

Collezioni documentali indicizzate e replica set per alta disponibilità.

DETtagli Backend & API

Il server Express funge da cuore applicativo, ricevendo richieste HTTP, applicando logica di autorizzazione e validazione, e delegando le operazioni sui dati a controller specializzati che interagiscono direttamente con MongoDB.

Middleware



CORS per origini cross-site, parsing JSON e validazione JWT prima di ogni chiamata ai router dedicati.

Router Modulari



File dedicati per /api/statistiche, /api/registro e /api/home che smistano le richieste ai controller appropriati.

Controller



Ogni controller incapsula la logica di una sezione dell'applicativo, orchestrando operazioni CRUD e gestione degli errori.

Connessione al Database



Modulo db/connection.js crea un'istanza unica del client MongoDB, condivisa da tutti i controller per eseguire query documentali efficienti.

DETtagli FRONTEND

Il client React è strutturato per isolare chiaramente la logica applicativa dallo stile e dal rendering, garantendo una navigazione protetta, gestione centralizzata dello stato e componenti UI riutilizzabili.



AppContext

Fornisce il contesto globale per autenticazione, memorizzando e distribuendo il token JWT e le informazioni dell'utente a tutta l'app.



ProtectedRoute

Wrapper che blocca l'accesso alle route sensibili (Registro) se non è presente un token valido, reindirizzando automaticamente al login.



Pagine Principali

Componenti "Page" di alto livello:

- Home (panoramica generale)
- Registro (dashboard studente/docente)
- Statistiche (con grafici e filtri)



Componenti UI Riutilizzabili

Elementi di base (Card, Button, Spinner, Accordion) isolati in file separati per garantire coesione e consistenza stilistica.

MODELLO DATI MONGODB

Il database MongoDB ospita quattro collezioni documentali corrispondenti alle entità chiave:

- classi: metadati di ogni classe e aggregati demografici;
- studenti: profili individuali con riferimento alla classe e attributi ESCS;
- docenti + assegnazioni: anagrafica insegnanti e relazioni classe-materia;
- voti: record di valutazione con riferimenti a studente, docente, tipologia e data.

Ogni collezione è indicizzata sulle chiavi primarie (id_classe, id_studente, ecc.) per garantire letture e aggregazioni rapide senza schema rigido.



TIMELINE PRESENTAZIONE

In questa sezione riepiloghiamo i principali risultati ottenuti e traiamo le conclusioni sull'efficacia del progetto, evidenziando punti di forza, limiti attuali e possibili sviluppi futuri.

Introduzione e Motivazioni

Use Case & Requisiti Utente

Definizione del Mini-world e Dataset

Pipeline ETL in Python

Modello di Calcolo dei Voti

Architettura Full-Stack

Risultati & Conclusioni

RISULTATI

Al termine dello sviluppo, NoSchoolQL dimostra di possedere una pipeline ETL robusta, grazie alla suddivisione in script indipendenti e all'uso di un seed fisso per la riproducibilità, e di un'architettura full-stack modulare, con un backend Express completamente stateless e un frontend React/Vite fluido e protetto.

Il modello dati NoSQL si è rivelato efficiente, supportando query rapide su classi, studenti, docenti e voti senza schema fisso, mentre i pesi socio-demografici e le difficoltà delle materie hanno mantenuto un perfetto allineamento con le evidenze ufficiali.



GRAZIE PER L'ATTENZIONE



ADG