Università degli Studi di Salerno

Dipartimento di Informatica

Laurea Triennale in Informatica

Corso: Machine Learning

STATUS

Automatic User Experience Level Classification from Synthetic Gym Training Logs

Autore:
Alessandro Ambrosio

Matricola:
0512119148

Anno Accademico:
A.A. 2025/2026

GitHub Repository

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni il settore del fitness sta vivendo una progressiva integrazione con strumenti digitali (app, wearable e servizi online), con un ruolo crescente per l'analisi dei dati e l'intelligenza artificiale nella personalizzazione dell'allenamento e nel monitoraggio dei progressi. In particolare, il mercato dell'online fitness mostra una crescita rilevante (da circa 36.64 miliardi USD nel 2026 a 120.13 miliardi USD nel 2031, con CAGR stimato 26.82%), a conferma di un contesto in espansione in cui soluzioni data-driven possono generare valore. Analogamente, anche il segmento dei fitness tracker risulta in forte crescita, indicando un'adozione sempre maggiore di tecnologie orientate al monitoraggio dell'attività fisica e della salute.

La scelta di sviluppare questo progetto nasce anche da un interesse personale per entrambe le aree, fitness e informatica, e vuole rappresentare un primo approccio concreto all'unione dei due ambiti: da un lato la logica dell'allenamento (carichi, frequenza, aderenza, progressione), dall'altro la costruzione di pipeline dati e modelli di machine learning in grado di estrarre segnali utili da tali informazioni. L'idea di fondo è che, in un mercato in crescita e sempre più orientato a soluzioni personalizzate, la capacità di trasformare dati di allenamento in insight e previsioni costituisca un passaggio chiave per migliorare l'esperienza dell'utente e la qualità del supporto decisionale.

In questo contesto, il lavoro si focalizza sul modulo **STATUS**, cioè un sistema di classificazione del livello di esperienza dell'utente (*Beginner*, *Intermediate*, *Advanced*) a partire da feature aggregate derivate da log di allenamento e indicatori di comportamento (es. aderenza/skip rate, volume complessivo, andamento del carico). L'obiettivo è duplice: (i) costruire un flusso completo e riproducibile (feature engineering $\rightarrow$ training $\rightarrow$ valutazione $\rightarrow$ salvataggio e reporting), (ii) analizzare criticamente le performance ottenute e i fattori che guidano le decisioni del modello (interpretazione tramite confusion

matrix e feature importance).

Il resto dell'elaborato è organizzato come segue: la Metodologia descrive dataset, feature engineering, modelli e protocollo di valutazione; i Risultati riportano le performance e le analisi (confusion matrix, metriche per classe, feature importance); Discussion e conclusioni discutono implicazioni, limiti e possibili sviluppi futuri.

Capitolo 2

Scopo, contesto e criteri di validità

2.1 Scopo del report (STATUS)

Questo elaborato documenta **esclusivamente** il modulo *STATUS*, un sistema di classificazione multi-classe che assegna a ciascun utente uno tra tre livelli di esperienza (*Beginner*, *Intermediate*, *Advanced*) a partire dai log di allenamento generati dal dataset sintetico descritto nel Chapter 2.

Fuori scopo. Il presente report non tratta la modellazione decisionale temporale o la stima di rischi/score continui; eventuali moduli aggiuntivi (es. IMPETUS) sono considerati, al più, come modulo parallelo e non sono necessari per comprendere o valutare STATUS.

Obiettivo scientifico. L'obiettivo è valutare (i) una pipeline end-to-end (preprocessing → feature engineering → training → valutazione), e (ii) l'efficacia predittiva su un contesto controllato e riproducibile, evitando ambiguità su dati, label e split.

2.2 Dataset sintetico e motivazione d'uso

Il dataset utilizzato è sintetico e deriva da un generatore set-level (v2.0) che simula: pianificazione sessioni, dropout (skip), esecuzione dei set (carico, ripetizioni, RPE), dinamiche di fatica/recupero e metadati utente. Le tabelle principali e lo schema relazionale sono descritti nel Chapter 3.

Motivazione. L'uso di dati sintetici consente di (i) controllare le distribuzioni dei fenomeni simulati, (ii) tracciare l'intero processo generativo, (iii) garantire replicabilità degli esperimenti, e (iv) isolare l'analisi ML da vincoli di privacy e disponibilità dati reali.

2.3 Definizione della label e del costrutto "esperienza"

In STATUS, la variabile target `experience_label` rappresenta un costrutto operativo di "livello di esperienza" coerente con pattern comportamentali e di allenamento simulati dal generatore (es. aderenza, calibrazione RPE, gestione del carico). Tale costrutto è definito rispetto alle ipotesi del modello generativo e non pretende di coincidere con una misurazione clinica o biomeccanica della performance.
Implicazione. La valutazione del modello misura la capacità di apprendere la relazione tra feature e label *nel dominio sintetico*; eventuali conclusioni su popolazioni reali richiederebbero una validazione esterna su dati osservati.

2.4 Minacce alla validità e mitigazioni (STATUS)

Questa sezione elenca in modo esplicito le principali minacce alla validità e le mitigazioni adottate, con l'obiettivo di rendere trasparente cosa può (e non può) essere inferito dai risultati.

2.4.1 Validità di costrutto

Minaccia. Possibile "slippage" tra il concetto informale di esperienza e l'implementazione operativa nel generatore (ovvero: la label potrebbe catturare solo alcune componenti del concetto).
Mitigazione. Il report:

- definisce esplicitamente il costrutto e le sue assunzioni (questa sezione);
- riporta EDA mirata a verificare che le feature siano informative ma non banali;
- include analisi di errore per comprendere casi di confine tra classi.

2.4.2 Validità interna (leakage e dipendenze tra campioni)

Minaccia. Con dati a granularità set/workout, esiste dipendenza tra osservazioni dello stesso utente; uno split casuale per-riga può introdurre contaminazione tra train e test (il modello "riconosce" l'utente).
Mitigazione. Nel modulo STATUS:

- lo split train/test viene eseguito a livello utente (group split), in modo che tutte le osservazioni di uno stesso `user_id` appartengano a un solo insieme;

- si evitano feature costruite direttamente tramite clamp/range deterministici per classe (o si documenta l'ablation per quantificarne l'impatto);
- le feature sono calcolate usando solo informazioni disponibili fino all'istante considerato (se si adottano finestre temporali).

2.4.3 Validità statistica (metriche e reporting)

Minaccia. Metriche aggregate possono nascondere prestazioni sbilanciate tra classi.
Mitigazione. Il report riporta:

- metriche per classe (precision/recall/F1) oltre all'accuracy;
- matrice di confusione e analisi degli errori principali;
- (opzionale) intervalli di confidenza tramite bootstrap sul test set.

2.4.4 Validità esterna (generalizzabilità)

Minaccia. Il dataset sintetico può non riflettere tutte le complessità del mondo reale (strumenti di tracking, comportamento, rumore di misura, variazioni inter-soggetto).
Mitigazione. Le conclusioni del report sono limitate al dominio sintetico; l'eventuale trasferimento su dati reali è presentato come lavoro futuro.

2.5 Riproducibilità e tracciabilità esperimenti

La generazione del dataset e gli esperimenti STATUS sono resi riproducibili tramite:

- seed fissato e RNG esplicito;
- salvataggio di metadati (versione, seed, dimensioni, range temporali, distribuzioni);
- versionamento del codice e (quando possibile) esportazione degli artefatti del modello (pipeline/feature list).

Capitolo 3

Dataset sintetico e generatore

3.1 Panoramica

Il presente progetto utilizza un dataset sintetico set-level generato mediante il supporto di *AI generativa*, progettato per simulare lo storico di allenamento di utenti di palestra. Il processo di generazione è stato strutturato per riprodurre in modo controllato le principali caratteristiche statistiche e dinamiche del dominio, includendo sia comportamento (skip/dropout), sia variabili di esecuzione (carichi, ripetizioni, RPE), sia dinamiche temporali (fatica, detraining, variazioni settimanali). L'obiettivo del dataset è fornire un contesto controllato e riproducibile per l'addestramento e la valutazione del modulo *STATUS*. Si precisa che le osservazioni non derivano da dati reali ma sono state create a fini sperimentali; pertanto, i risultati devono essere interpretati nel contesto di dati sintetici, con le relative limitazioni in termini di generalizzabilità a scenari reali. Le assunzioni e le minacce alla validità collegate all'uso di dati sintetici sono discusse nel Chapter 2.

3.2 Fondamenti scientifici del generatore

Il generatore è progettato per produrre pattern temporali e comportamentali plausibili, ispirandosi a modelli e risultati ricorrenti nella letteratura su carico di allenamento, adattamento e aderenza.

3.2.1 Fitness–Fatigue (Impulse–response)

L'andamento di performance/condizione nel tempo può essere modellato come risposta a impulsi di training con due componenti antagoniste: una componente positiva (fitness)

e una negativa (fatigue), entrambe con decadimento esponenziale. [1] Questo razionale è coerente con la famiglia di modelli Banister/Calvert e con l'uso di strutture a due stati (fitness e fatigue) come base concettuale per simulare cicli di overload e recupero. [2], [3]

3.2.2 ACWR e rilevazione di spike

Il rapporto Acute:Chronic Workload Ratio (ACWR) è utilizzato come indicatore sintetico di carico acuto rispetto alla preparazione cronica, e spike (incrementi rapidi) sono spesso associati a incremento di rischio di infortunio in letteratura su team sports. [4] Nel generatore, una soglia operativa di spike pari a `ACWR > 1.61` viene adottata per marcare settimane ad alto carico relativo, in linea con la logica di evitare aumenti rapidi e con discretizzazioni che collocano ~ 1.6 al confine superiore della fascia moderata-alta. [4]

3.2.3 Progressive overload e plateau per esperienza

Il principio di progressive overload è implementato imponendo tassi di crescita differenti per livello di esperienza (novizi con crescita più rapida, avanzati con crescita più lenta), simulando sia adattamenti iniziali più veloci sia fenomeni di plateau con l'aumentare dell'anzianità di allenamento.

3.2.4 RPE e calibrazione (RIR-based)

La Rating of Perceived Exertion (RPE) viene trattata come misura di carico interno, con rumore/errore di stima potenzialmente dipendente dall'esperienza dell'utente. [5] Il generatore si ispira all'approccio RPE basato su Repetitions In Reserve (RIR) per rendere la variabile più interpretabile e coerente con un contesto di resistance training. [5]

3.2.5 Pattern di dropout e aderenza

L'aderenza nei contesti non supervisionati (es. palestra) tende a decrescere nel tempo e la frequenza di allenamento è un forte indicatore di dropout; per questo nel generatore sono presenti eventi `skipped` e profili con rischio diverso. [6]

3.3 Riproducibilità e versionamento

La generazione è parametrizzata da una configurazione (`CFG`) e da un seed fissato (`CFG.seed`).Per evitare dipendenze da stato globale e garantire replicabilità, il generatore istanzia un RNG moderno e lo propaga alle funzioni di campionamento (pattern *pass-the-RNG*).I metadati di generazione (seed, dimensioni, intervalli temporali, distribuzioni aggregate, changelog) sono salvati in `metadata_v2.json`.

Nota implementativa. Nel codice Python si adotta `np.random.default_rng(seed)` per costruire un oggetto `Generator` e campionare in modo deterministico a seed fissato.

3.4 Tabelle esportate e granularità

Il dataset è organizzato in tabelle CSV correlate; la granularità varia da *utente* a *sessione* a *set* fino a *giorno* (feature derivate).

Tabella 3.1: File esportati e contenuto.

File	Descrizione (chiavi principali)
`users.csv`	1 riga/utente (`user_id`); profilo, label, tratti, finestra temporale
`exercises.csv`	catalogo esercizi (`exercise_id`); split e gruppi muscolari
`workouts.csv`	1 riga/sessione; stato `done/skipped`, probabilità skip, tag split
`workout_plan.csv`	prescrizione per esercizio in sessione; set pianificati, reps range, rest, RIR
`workout_sets.csv`	**primary**; 1 riga/set (`set_id`); carico, reps, RPE, ACWR, `workout_id`
`banister_meta.csv`	parametri Banister per utente (`user_id`): $\tau_F, \tau_D, \beta_F, \beta_D$
`banister_daily.csv`	1 riga/giorno per utente; fitness, fatigue, TSB, performance (0–100)
`metadata_v2.json`	metadati: seed, dimensioni, distribuzioni, changelog

3.5 Configurazione experience-aware

La versione v2.0 introduce una parametrizzazione *experience-aware* (dipendente da `experience_label`) per modellare pattern comportamentali plausibili e differenze sistematiche tra livelli.I parametri principali includono: skip rate di base, calibrazione RPE (bias e varianza), probabilità di settimane spike/deload, durata dello storico, e range di consistency.

Tabella 3.2: Parametri experience-aware principali (v2.0).

Parametro	**Beginner**	**Intermediate**	**Advanced**
Skip p_0 (baseline)	0.13	0.075	0.05
Skip fatigue sensitivity	0.35	0.25	0.15
RPE noise std	1.5	1.0	0.6
RPE bias	+0.5	0.0	-0.2
Spike prob (settimana)	0.20	0.12	0.06
Deload prob (settimana)	0.03	0.05	0.08
Durata storico (giorni)	90–270	240–600	540–1080
Consistency target	0.65–0.80	0.75–0.90	0.85–0.95

3.6 Processo generativo (utente → sessioni → set)

La simulazione è eseguita per utente e scorre le sessioni pianificate tra `start_date` e `end_date`.Ogni sessione può essere completata o saltata (skip), e se completata genera un piano esercizi e una sequenza di set eseguiti.

3.6.1 Pianificazione sessioni

Le date di allenamento sono generate in base a una frequenza settimanale dichiarata (`weekly_freq_declared`) e a un jitter sul giorno della settimana per introdurre variabilità realistica.

3.6.2 Modello di skip (session-level)

La probabilità di skip p_{skip} è definita come una trasformazione logistica della probabilità base per livello p_0 (modulata dalla disciplina utente) e di un termine di fatica saturato:

$$p_{\text{skip}} = \sigma\Big(\text{logit}(p_0) + w_{\text{fat}} \cdot \log(1 + \text{fatigue}) + \epsilon\Big),$$

dove w_{fat} dipende dal livello e ϵ è rumore gaussiano. I campi `p_skip`, `z_skip` e `fatigue_term` sono salvati in `workouts.csv` per consentire audit e analisi.

3.6.3 ACWR settimanale e week type

A inizio di ciascuna settimana utente, viene calcolato un moltiplicatore ACWR che determina il tipo settimana `normal/spike/deload` e influenza l'ampiezza dell'allenamento (es. set eseguiti).Il tipo settimana è salvato a livello set (`week_type`) insieme al valore `acwr`.

3.6.4 Esecuzione set: load, reps, RPE e missingness

Per ciascun esercizio pianificato, il numero di set eseguiti è una funzione dei set pianificati, del profilo utente (`profile`) e del moltiplicatore settimanale.Ogni set produce:

- `load_intended_kg` e `load_done_kg` (con quantizzazione e rumore osservazionale);
- `reps_target` e `reps_done` (con degradazione dovuta alla fatica intra-sessione);
- `rpe_done` (osservato) ottenuto da una componente "true" e da un errore di reporting (bias+varianza) dipendente dal livello.

Per aumentare realismo, è introdotta missingness controllata su `load_done_kg`, `rpe_done` e `feedback`.

3.7 Feature day-level: modello di Banister

A partire dall'impulso giornaliero (`impulse`), viene calcolata una serie temporale day-level tramite un modello impulso-risposta: fitness e fatigue sono ottenute come convoluzione dell'impulso con kernel esponenziali, parametrizzati per utente da τ_F e τ_D.La quantità `TSB` è definita come differenza tra fitness e fatigue, e la variabile `performance` è una trasformazione scalata in [0,100] per interpretabilità.

3.8 Consistency score (post-processing)

La consistency è calcolata come rapporto tra giorni di allenamento completati e giorni totali nella finestra osservata, quindi viene riportata in un intervallo target dipendente dal livello, con lo scopo di evitare distribuzioni degeneri (es. consistency sempre pari a 1.0).

3.9 Validazioni e vincoli di qualità

Prima del salvataggio vengono eseguiti controlli automatici per garantire coerenza minima del dataset:

- integrità delle chiavi (`user_id` presente, `workout_id` non nullo nei set);
- vincoli su range (`load_done_kg` in [2.5, 200], `rpe_done` in [1,10], `reps_done` in [1,50]);
- presenza delle 3 classi `experience_label` nel dataset finale;
- stima e verifica della skip rate aggregata per livello.
- `consistency_score` è generata in modo deterministico per livello e viene usata solo per audit descrittivo; non è inclusa tra le feature di training per evitare target leakage.

3.10 Subset stratificato per sviluppo (STATUS)

Per velocizzare EDA e iterazioni di feature engineering, viene creato un subset stratificato di utenti bilanciato per classe (es. 170 utenti per livello, 510 totali).Il subset include versioni filtrate delle tabelle principali (suffix `_sampled`) e viene esportato anche in formato ZIP per uso locale.

Capitolo 4

Formulazione del problema

4.1 Contesto e motivazione

Questo progetto nasce dall'interesse personale per il fitness e per l'informatica e rappresenta un primo approccio concreto all'unione dei due ambiti: applicare metodi di data science e machine learning a dati di allenamento per ottenere insight e supportare decisioni (es. classificazione del profilo utente e personalizzazione).

4.2 Problema e obiettivo

Problema. Dati log di allenamento e indicatori derivati, stimare automaticamente un profilo/level dell'utente che sia utile per supportare raccomandazioni e analisi. **Obiettivo.** Progettare e validare una pipeline end-to-end per la stima del livello utente come task supervisionato, includendo feature engineering, training di modelli, valutazione e interpretazione.

4.3 Assunzioni e vincoli

Si assume che le feature ingegnerizzate catturino in modo sufficiente pattern di comportamento/allenamento discriminanti e che la definizione delle classi sia coerente con l'uso previsto del sistema. Il modello deve generalizzare a utenti non visti e mantenere un livello di overfitting controllato (gap train-test contenuto).

4.4 Criteri di successo

Il sistema è considerato soddisfacente se ottiene performance robuste sul test set e se gli errori residui sono interpretabili (es. confusione tra classi adiacenti), permettendo di motivare la scelta del modello e delle feature.

Capitolo 5

Exploratory Data Analysis (EDA)

5.1 Obiettivi dell'EDA

Questa sezione presenta l'Exploratory Data Analysis (EDA) del dataset sintetico per il modulo STATUS. L'obiettivo è (i) profilare il dataset (dimensioni, distribuzioni, qualità), (ii) verificare la separabilità tra classi del target, e (iii) motivare in modo trasparente le scelte di Feature Engineering e la selezione di feature per il training.

5.2 Profilazione del dataset

L'EDA è condotta sul subset stratificato (510 utenti, 170 per classe), costruito a partire dal dataset completo. Le tabelle principali utilizzate sono `users_sampled.csv`, `workouts_sampled.csv` e `workout_sets_sampled.csv`.

Tabella 5.1: Dimensioni del subset utilizzato per l'EDA di STATUS (v2.0).

Oggetto	**Conteggio**
Utenti	510
Workouts	106,571
Set	1,566,944

5.3 Distribuzione del target

Il target `experience_label` risulta bilanciato per costruzione (campionamento stratificato: 170 utenti per classe).

Tabella 5.2: Distribuzione classi del target `experience_label`.

Classe	Conteggio	%
Beginner	170	33.3
Intermediate	170	33.3
Advanced	170	33.3

5.4 Aggregazioni user-level

Poiché STATUS opera a livello utente, i dati set-level sono aggregati per `user_id` producendo feature descrittive (medie, deviazioni standard e conteggi) e feature derivate (es. `skip_rate`, `load_progression`). Inoltre, `spike_weeks_count` è definita come il numero di settimane uniche marcate come `spike` per ciascun utente.

5.5 Statistiche descrittive per classe

Questa sezione riassume le principali differenze tra classi in termini di comportamento e pattern di allenamento. Come misura sintetica di separazione tra gruppi, viene riportato Cohen's d per il confronto Beginner vs Advanced, con attenzione ai casi degeneri (varianza prossima a zero).

Tabella 5.3: Statistiche (media ± std) per classe e separazione (Cohen's d, Beginner vs Advanced).

Feature	Beginner	Intermediate	Advanced	d (B vs A)
load_mean	19.70 ± 1.57	33.45 ± 5.66	68.69 ± 23.55	2.935
reps_mean	8.96 ± 0.10	8.72 ± 0.09	8.68 ± 0.06	3.302
rpe_mean	5.10 ± 0.23	4.73 ± 0.20	4.59 ± 0.22	2.211
total_sets	723.41 ± 358.38	2451.34 ± 1371.33	6042.58 ± 2857.48	2.612
acwr_mean	1.20 ± 0.07	1.14 ± 0.04	1.09 ± 0.02	2.162
spike_weeks_count	6.85 ± 3.83	9.82 ± 4.90	9.75 ± 5.03	0.650
load_progression	1.01 ± 0.01	1.05 ± 0.06	1.18 ± 0.23	1.000
skip_rate	0.214 ± 0.053	0.116 ± 0.029	0.070 ± 0.017	3.657
observed_freq_weekly	3.13 ± 0.81	3.11 ± 0.84	3.16 ± 0.76	0.041

5.6 Correlazioni e controllo leakage

Per individuare possibili segnali di leakage o scorciatoie, viene analizzata la matrice di correlazione tra feature numeriche e una codifica ordinale del target. La variabile `consistency_score` risulta deterministica per classe (varianza nulla intra-gruppo) e viene quindi considerata non idonea come feature di training; nel seguito è usata solo per audit descrittivo.

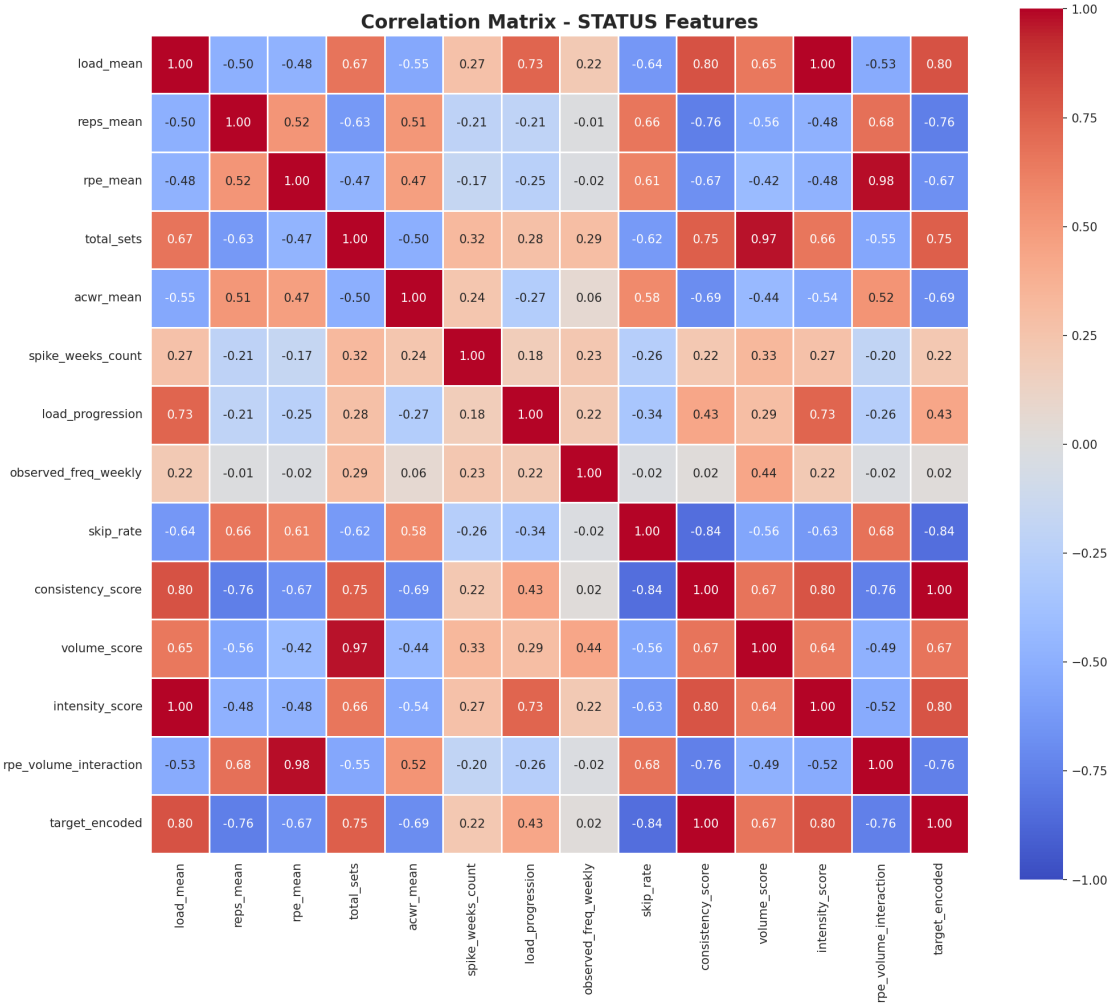

Figura 5.1: Matrice di correlazione tra feature user-level e target codificato.

5.7 Distribuzioni e visualizzazioni

Per supportare l'analisi descrittiva vengono riportate (i) distribuzioni per classe (istogrammi), (ii) densità (violin plots) e (iii) confronto Beginner vs Advanced (KDE).

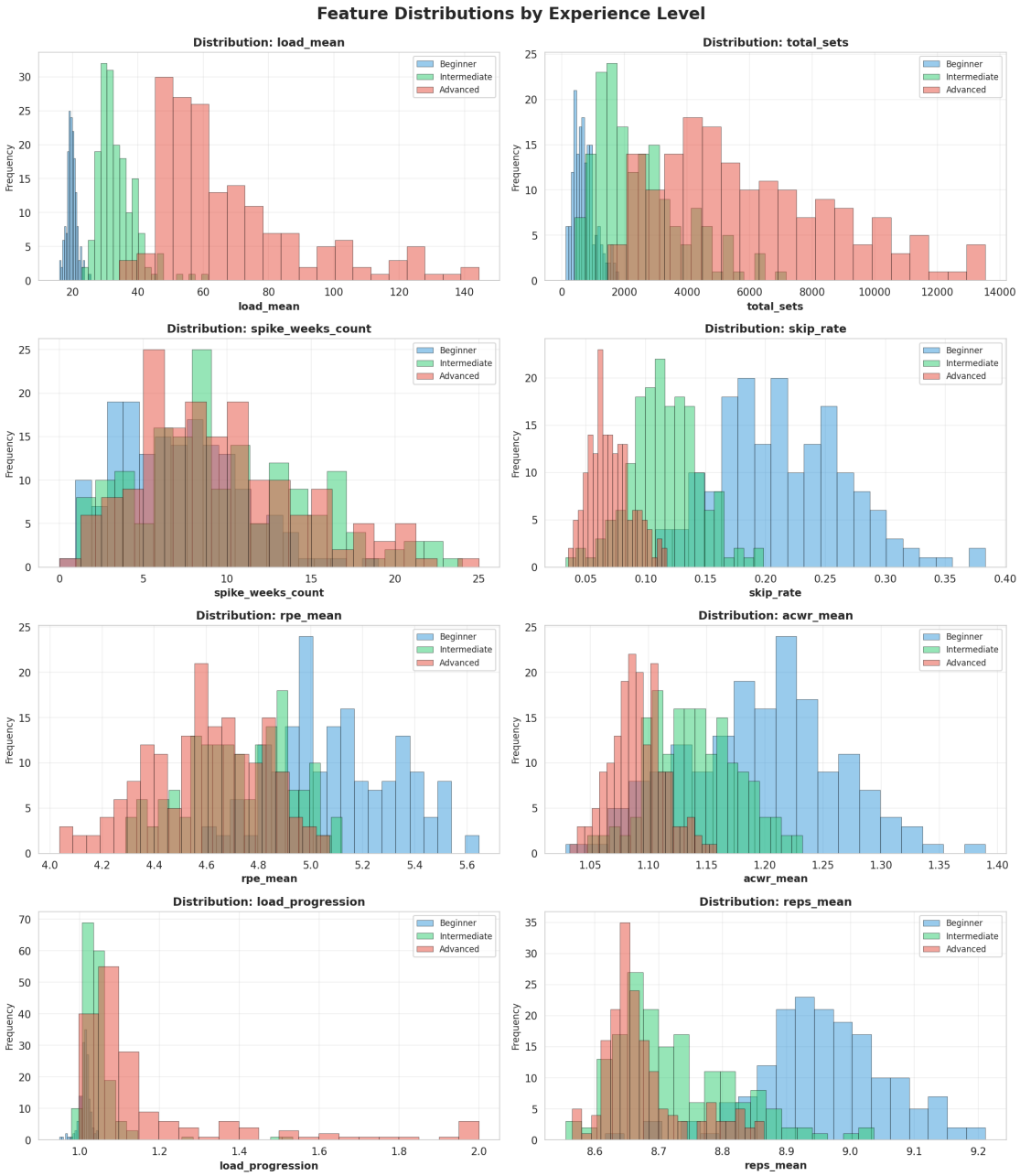

Figura 5.2: Istogrammi delle feature principali stratificati per `experience_label`.

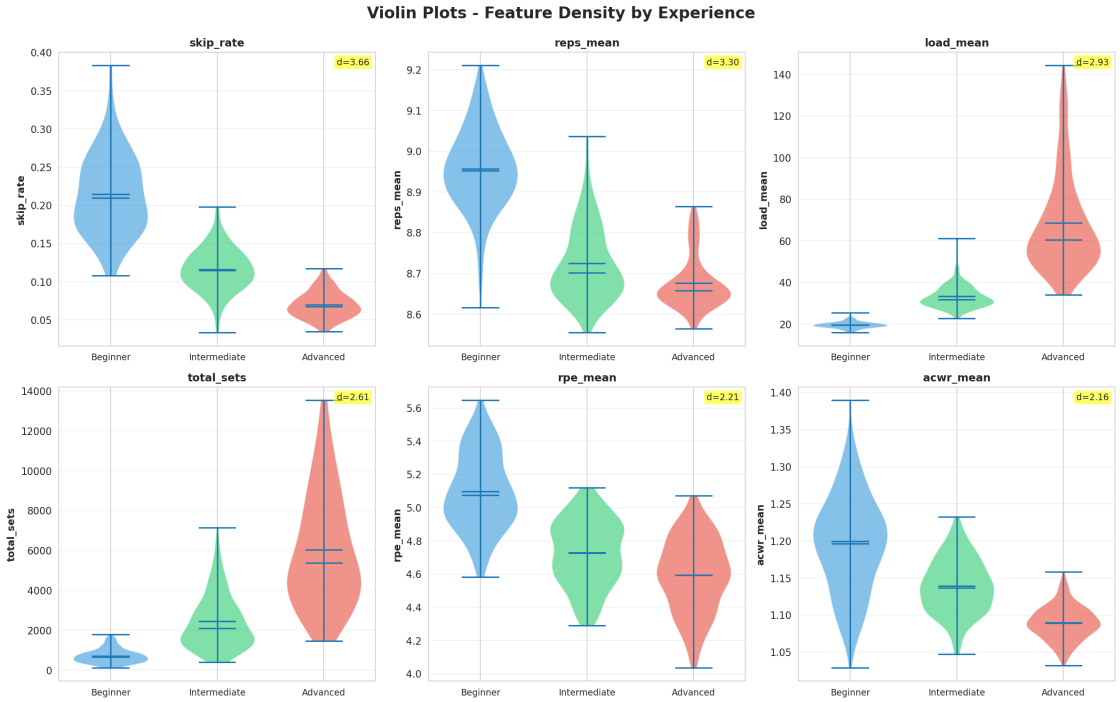

Figura 5.3: Violin plots delle feature più discriminanti.

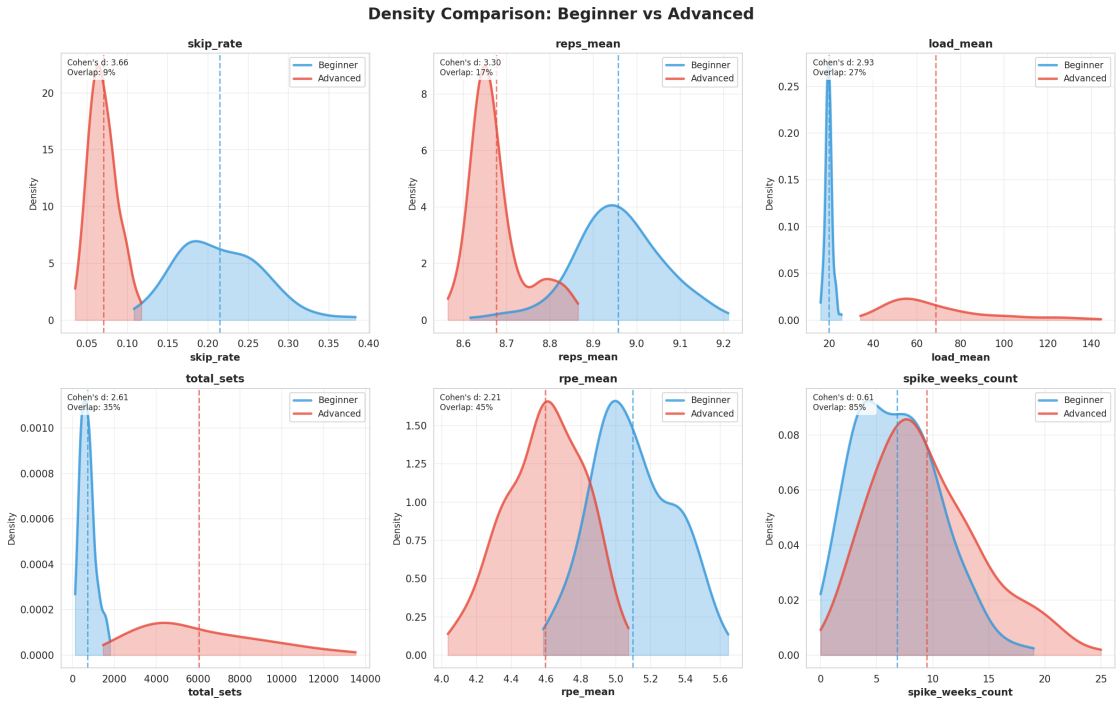

Figura 5.4: Confronto di densità (KDE) tra Beginner e Advanced per feature selezionate.

5.8 Pattern categoriali (workout behavior)

Infine, vengono analizzati pattern categoriali e di comportamento: distribuzione del `week_type`, statistiche della `skip_rate` e distribuzione binned di `total_sets`.

Tabella 5.4: Distribuzione `week_type` per classe (%).

Classe	Deload	Normal	Spike
Advanced	5.4	86.1	8.4
Beginner	2.0	71.5	26.5
Intermediate	3.6	79.9	16.5

Figura 5.5: Distribuzione di `week_type` per classe.

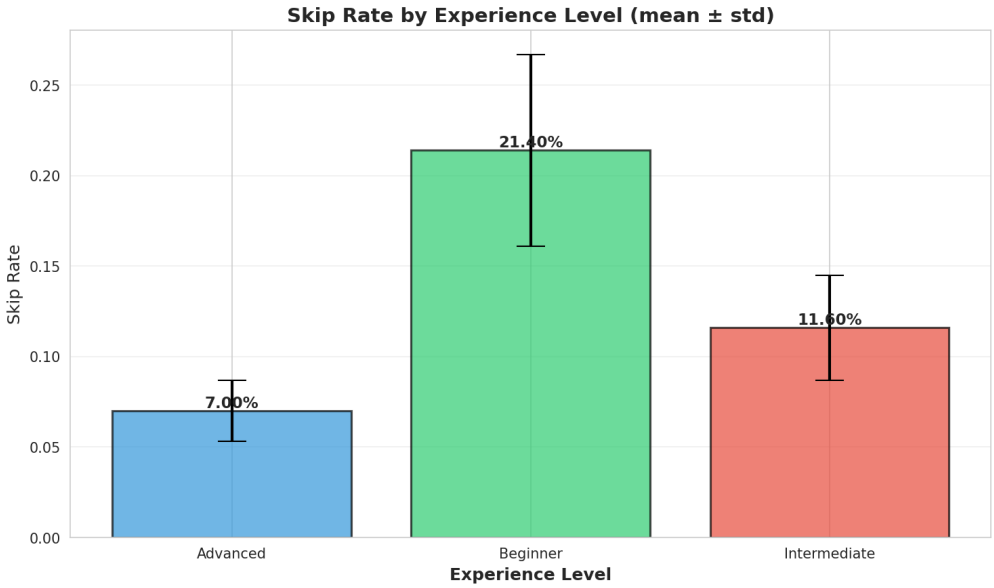

Figura 5.6: Skip rate per classe (media ± std).

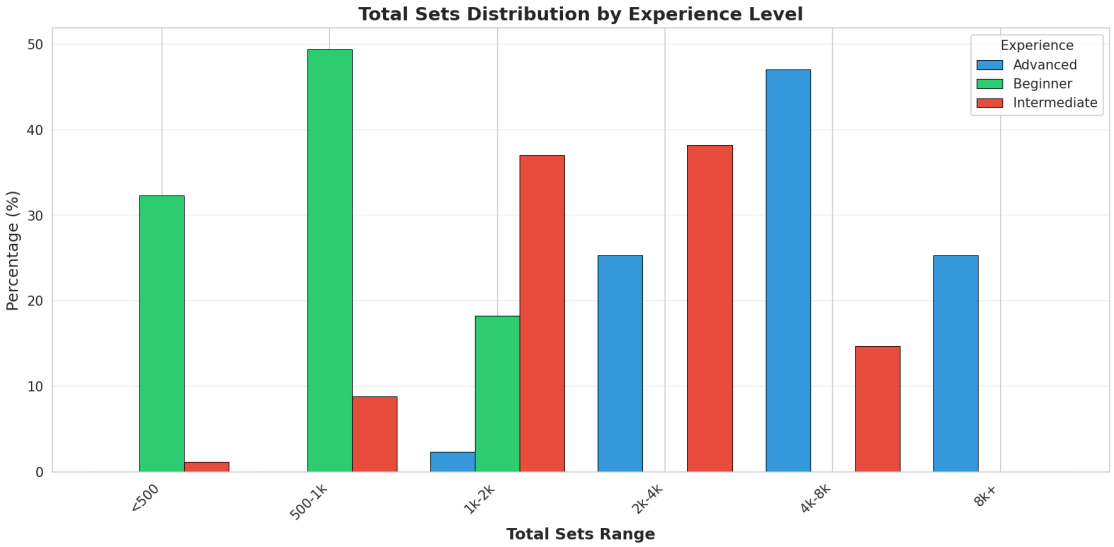

Figura 5.7: Distribuzione di `total_sets` (binned) per classe.

5.9 Output dell'EDA e implicazioni per la FE

Dall'EDA emergono feature fortemente discriminanti (es. `skip_rate`, `reps_mean`, `load_mean`) e feature deboli (es. `observed_freq_weekly`). Le scelte di Feature Engineering e la lista finale di feature per il training vengono motivate e formalizzate nel capitolo successivo.

Capitolo 6

Feature Engineering

6.1 Obiettivo e input/output

Questo capitolo descrive la pipeline di Feature Engineering (FE) per il modulo STATUS, che costruisce un dataset user-level per classificare `experience_label` (Beginner, Intermediate, Advanced). La FE utilizza il subset stratificato (510 utenti) e produce un bundle preprocessato contenente `X_train`, `X_test`, `y_train`, `y_test` e lo scaler per il riutilizzo nel notebook di modeling.

Tabella 6.1: Input e output della pipeline FE.

Tipo	**Path**
Input	data/synth_set_level_v2/users_sampled.csv
Input	data/synth_set_level_v2/workout_sets_sampled.csv
Input	data/synth_set_level_v2/workouts_sampled.csv
Output	models/status_preprocessed_v2.2.pkl
Output	models/status_feature_metadata_v2.2.json

6.2 Aggregazioni user-level

La pipeline trasforma i dati set-level e workout-level in una singola riga per utente tramite aggregazioni per `user_id`. Le feature base includono statistiche di carico, ripetizioni, RPE e un conteggio del volume complessivo (`total_sets`). Sono inoltre calcolate feature temporali/sintetiche come `acwr_mean`, `skip_rate` e `load_progression` (pendenza del carico nel tempo stimata via regressione lineare).

6.3 Definizione di `spike_weeks_count`

Per evitare un artefatto comune, `spike_weeks_count` è definita come il *numero di settimane uniche* marcate come `spike` per utente, e non come conteggio dei set appartenenti a settimane spike. Operativamente, a ciascun set viene associato un identificatore di settimana (`week_id`) e la feature viene calcolata come `nunique` delle settimane spike per `user_id`. Questa scelta rende la feature confrontabile tra utenti con diversi volumi di set e impedisce una sovrastima puramente guidata da `total_sets`.

6.4 Selezione feature e controllo leakage

La lista finale di feature è composta da 7 variabili, selezionate per buona separabilità tra classi e rischio di leakage contenuto. Sono rimosse: (i) `consistency_score` perché deterministica per classe (leakage perfetto), (ii) `load_mean` per rischio di shortcut/borderline leakage, e (iii) `observed_freq_weekly` per bassa potenza discriminativa. Queste scelte sono tracciate nel file di metadata (`status_feature_metadata_v2.2.json`) per garantire riproducibilità e interpretabilità.

Tabella 6.2: Feature selezionate per STATUS.

Feature	**Descrizione sintetica**
reps_mean	Ripetizioni medie per set
rpe_mean	RPE medio (intensità percepita)
total_sets	Volume storico totale (conteggio set)
acwr_mean	Indicatore medio di carico acuto/cronico
spike_weeks_count	Numero settimane spike per utente
load_progression	Tasso di progressione del carico (mensile, normalizzato)
skip_rate	Frazione di workout `skipped`

6.5 Split train/test e scaling

Il dataset user-level viene suddiviso in train/test (80/20) con stratificazione su `experience_label`, preservando la distribuzione delle classi tra le partizioni. Per uniformare il preprocessing tra modelli, viene applicato `StandardScaler` a tutte le feature numeriche: lo scaler viene **fit** esclusivamente sul train e poi usato per trasformare

sia train sia test, riducendo il rischio di data leakage e mantenendo una pipeline consistente. **scikit-common-pitfalls**

6.6 Artefatti salvati per il modeling

Il file `status_preprocessed_v2.2.pkl` contiene le matrici preprocessate (scalate) e lo scaler, permettendo di replicare il preprocessing in fase di training e di inferenza. Il file `status_feature_metadata_v2.2.json` riporta versione, lista feature, motivazioni di rimozione e informazioni sullo split, facilitando audit e interpretabilità.

Capitolo 7

Metodologia sperimentale

7.1 STATUS – Model training and evaluation protocol

Per il modulo STATUS è stato impostato un task di classificazione multiclasse a 3 livelli (Beginner, Intermediate, Advanced) utilizzando un set di 7 feature aggregate per utente: `reps_mean`, `rpe_mean`, `total_sets`, `acwr_mean`, `spike_weeks_count`, `load_progression`, `skip_rate`.

Il dataset contiene 510 utenti ed è stato suddiviso in training set e test set con split 80/20 (408/102); la suddivisione è stata stratificata per garantire la stessa distribuzione delle classi tra train e test.

Sono stati confrontati sei modelli: Dummy Classifier (baseline stratified), Logistic Regression, Decision Tree, Random Forest, Gradient Boosting e XGBoost. La valutazione è stata eseguita sul test set tramite Accuracy e F1-macro; inoltre è stato calcolato il train-test gap (Accuracy train – Accuracy test) come controllo di overfitting.

Capitolo 8

Risultati

8.1 STATUS – Model comparison and selected model

La Tabella 8.1 riporta le performance dei modelli confrontati sul test set. Il modello con performance migliore risulta Gradient Boosting, con Accuracy pari a 0.951 e F1-macro pari a 0.951, mantenendo un gap train-test di 0.049 (generalizzazione adeguata rispetto alla soglia fissata a 0.10).

Tabella 8.1: Confronto modelli (test set) – STATUS

Modello	**Acc (test)**	**F1-macro (test)**	**Gap**	**Time (s)**
Dummy (Stratified)	0.294	0.295	0.074	0.002
Logistic Regression	0.931	0.931	0.037	0.011
Decision Tree	0.912	0.912	0.032	0.009
Random Forest	0.941	0.941	0.044	0.604
Gradient Boosting	0.951	0.951	0.049	1.979
XGBoost	0.931	0.931	0.069	0.536

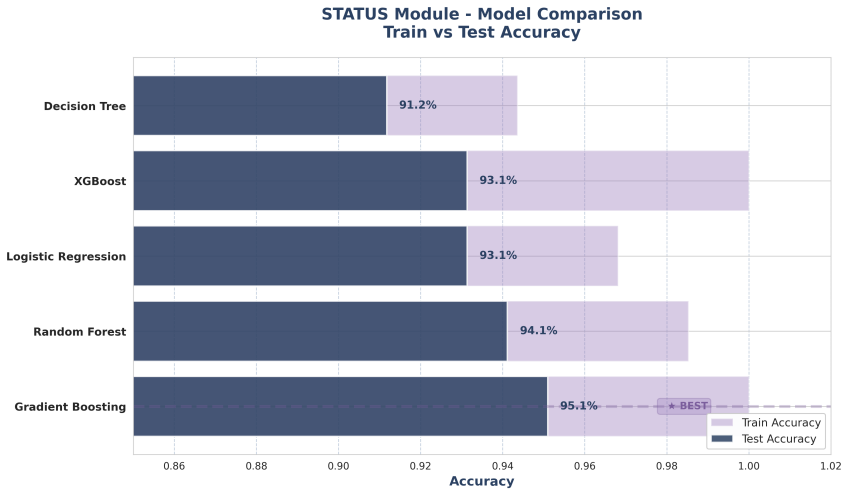

Figura 8.1: Confusion matrix del modello selezionato (Gradient Boosting) sul test set.

La Confusion Matrix (Figura 8.2) evidenzia che gli errori principali avvengono tra le classi *Intermediate* e *Advanced*. Per definizione, l'elemento $C_{i,j}$ della confusion matrix indica il numero di osservazioni della classe vera i predette come classe j. **sklearn_confusion_matrix**

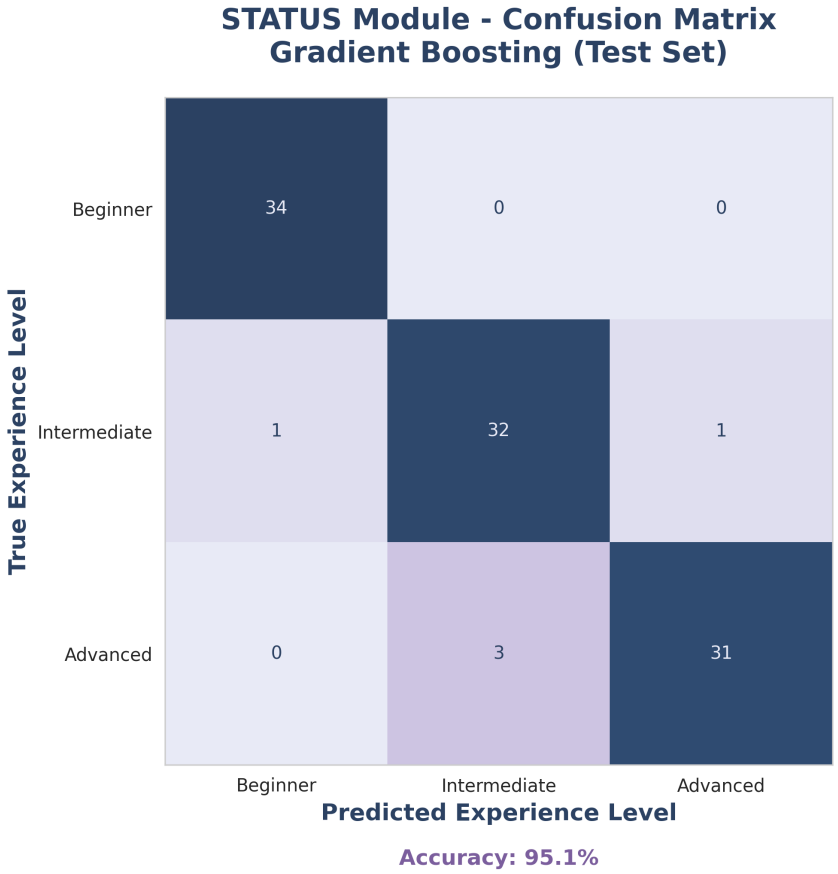

Figura 8.2: Confusion matrix del modello selezionato (Gradient Boosting) sul test set.

Le metriche per classe sul test set sono: Beginner (Precision 0.971, Recall 1.000, F1 0.986), Intermediate (Precision 0.914, Recall 0.941, F1 0.928), Advanced (Precision 0.969, Recall 0.912, F1 0.939).

Feature importance

L'analisi dell'importanza delle feature del modello Gradient Boosting (Figura 8.3) mostra una forte dominanza di `skip_rate` (0.615, pari al 61.46% del totale). Le prime tre feature (`skip_rate`, `reps_mean`, `total_sets`) coprono complessivamente l'87.89% dell'importanza totale, indicando che la regolarità/aderenza all'allenamento costituisce il segnale maggiormente discriminante nel dataset.

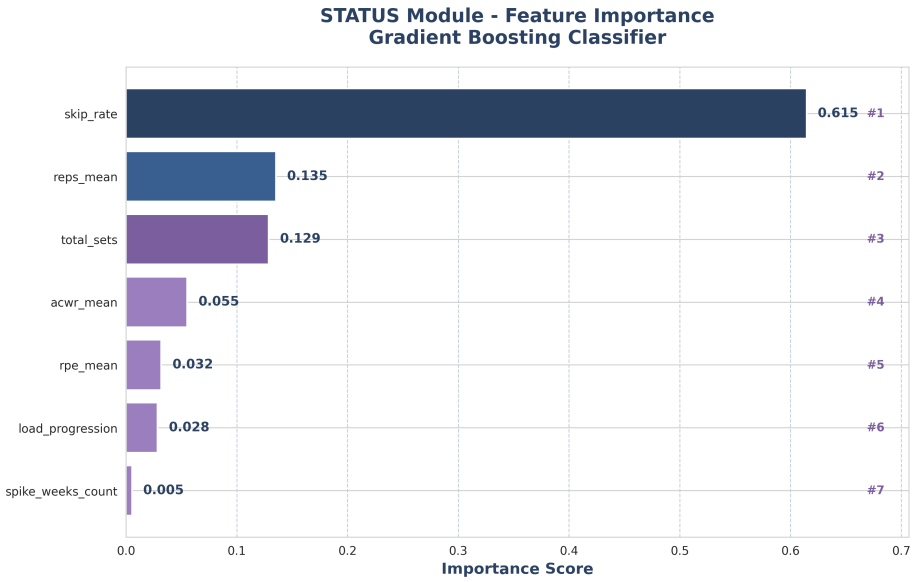

Figura 8.3: Feature importance del Gradient Boosting (test set).

Capitolo 9

Discussione

I risultati del modulo STATUS mostrano che, con le feature ingegnerizzate, modelli relativamente standard di classificazione riescono a separare in modo efficace i tre livelli di esperienza. Il modello selezionato (Gradient Boosting) ottiene performance elevate sul test set e un train-test gap contenuto, suggerendo una buona capacità di generalizzazione nel contesto dei dati utilizzati. L'analisi della confusion matrix evidenzia che gli errori residui si concentrano principalmente tra *Intermediate* e *Advanced*, cioè tra classi adiacenti e naturalmente più difficili da separare: questo comportamento è coerente con l'idea che i profili borderline condividano pattern di allenamento simili.

Un aspetto particolarmente importante emerge dalla feature importance: il modello risulta fortemente guidato dalla `skip_rate`, seguita da indicatori di volume e comportamento in sessione (es. `reps_mean` e `total_sets`). Questa dominanza può essere letta in modo positivo (aderenza e costanza sono effettivamente correlate all'esperienza e alla strutturazione dell'allenamento), ma richiede anche una riflessione critica: se una singola feature domina la decisione, il modello potrebbe diventare sensibile a rumore o bias nella generazione/registrazione di quell'indicatore. In altre parole, il classificatore potrebbe essere molto bravo a riconoscere la regolarità dell'utente più che la qualità tecnica o la capacità prestativa, che in un contesto reale potrebbero richiedere segnali diversi (es. progressioni di carico più dettagliate, intensità relativa, esercizi chiave, misure antropometriche o performance test).

Dal punto di vista applicativo, l'approccio resta comunque utile: una stima affidabile del livello di esperienza può supportare raccomandazioni iniziali (scelta di routine, range di volume/intensità, gestione dei picchi di carico) e migliorare l'onboarding dell'utente. Allo stesso tempo, questi risultati rafforzano l'idea che progetti di data science in ambito fitness possano integrare competenze di dominio (programmazione dell'allenamento)

e competenze tecniche (modellazione, pipeline, valutazione) in un workflow unico e replicabile, che costituisce un primo passo verso sistemi più completi di coaching digitale.

Capitolo 10

Limitazioni e sviluppi futuri

10.1 Limitazioni

La principale limitazione del progetto riguarda la natura dei dati: l'uso di un dataset sintetico, pur utile per progettare e testare l'intera pipeline end-to-end, non garantisce che le distribuzioni e le correlazioni tra feature riflettano pienamente la complessità del mondo reale. In particolare, la forte importanza di `skip_rate` potrebbe essere in parte amplificata dalle assunzioni con cui sono stati generati i dati, e potrebbe ridursi (o cambiare) su utenti reali con comportamenti più eterogenei.

Una seconda limitazione riguarda la rappresentazione del concetto di "esperienza": la label *Beginner/Intermediate/Advanced* è una semplificazione di un costrutto multidimensionale (tecnica, forza relativa, anzianità di training, qualità della programmazione, consistenza, obiettivi). Questo implica che un modello ottimizzato su una definizione specifica di "esperienza" potrebbe non trasferirsi bene a contesti in cui l'esperienza è definita in modo diverso (es. sport-specific, strength vs hypertrophy, ecc.).

Infine, la valutazione è basata su un singolo split train/test; sebbene lo split sia bilanciato e le metriche siano solide, una validazione più robusta (es. cross-validation ripetuta) ridurrebbe ulteriormente la varianza della stima prestazionale e migliorerebbe l'affidabilità dei confronti tra modelli.

10.2 Sviluppi futuri

Un primo sviluppo naturale consiste nel validare lo stesso framework su dati reali (o semi-reali) e introdurre controlli più stringenti sulla qualità delle misure (missingness, rumore, outlier comportamentali). In parallelo, si possono ampliare le feature per

catturare meglio la componente “prestativa” dell’esperienza: trend temporali di carico per esercizi chiave, indicatori di intensità relativa, metriche di progressione per pattern di movimento, e misure di fatica/recupero.

Dal punto di vista modellistico, sono possibili diversi miglioramenti: (i) validazione con cross-validation stratificata, (ii) calibrazione delle probabilità per rendere l’output interpretabile come confidenza, (iii) metodi di interpretabilità più robusti (es. permutation importance/SHAP) per verificare la stabilità delle feature importanti, (iv) modelli temporal-aware (es. sequenziali) se in futuro si decide di utilizzare direttamente serie temporali invece di sole aggregazioni per utente.

Infine, sul piano applicativo, l’evoluzione del progetto potrebbe includere un’integrazione in una demo interattiva (dashboard o app) con spiegazioni user-friendly del perché della classificazione, e con raccomandazioni coerenti con il profilo stimato (progressioni conservative per beginner, gestione del carico e del rischio di spike per intermediate/advanced).

Bibliografia

[1] F. Imbach, N. Sutton-Charani, J. Montmain, R. Candau e S. Perrey, «The Use of Fitness-Fatigue Models for Sport Performance Modelling: Conceptual Issues and Contributions from Machine-Learning,» *Sports Medicine - Open*, vol. 8, n. 29, 2022. DOI: 10.1186/s40798-022-00426-x indirizzo: https://pmc.ncbi.nlm.nih.gov/articles/PMC8894528/

[2] T. W. Calvert, E. W. Banister, M. V. Savage e T. Bach, «A Systems Model of the Effects of Training on Physical Performance,» *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 94–102, 1976. DOI: 10.1109/TSMC.1976.5409179 indirizzo: https://pmc.ncbi.nlm.nih.gov/articles/PMC8894528/

[3] E. Banister, T. Calvert, M. Savage e T. Bach, «A Systems Model of Training for Athletic Performance,» *Australian Journal of Sports Medicine*, vol. 7, n. 3, pp. 57–61, 1985. indirizzo: https://pmc.ncbi.nlm.nih.gov/articles/PMC8894528/

[4] L. Bowen, A. S. Gross, M. Gimpel, S. Bruce-Low e F.-X. Li, «Spikes in acute:chronic workload ratio (ACWR) associated with a 5–7 times greater injury rate in English Premier League football players: a comprehensive 3-year study,» *British Journal of Sports Medicine*, vol. 54, n. 12, pp. 731–738, 2019. DOI: 10.1136/bjsports-2018-099422 indirizzo: https://pmc.ncbi.nlm.nih.gov/articles/PMC7285788/

[5] E. R. Helms, J. Cronin, A. Storey e M. C. Zourdos, *Application of the Repetitions in Reserve-Based Rating of Perceived Exertion Scale for Resistance Training*, PMCID: PMC4961270, 2016. indirizzo: https://pmc.ncbi.nlm.nih.gov/articles/PMC4961270/

[6] «Dropout predictors at gyms: a retrospective study,» *Revista Brasileira de Ciência e Movimento (SciELO)*, 2021. DOI: 10.1590/rbce.43.e014220 indirizzo: https://www.scielo.br/j/rbce/a/WtzM3gBFRkcrqY7xKZsVNnm/?lang=en

Appendice A

Dettagli implementativi

.1 Output del generatore

Tabella 1: File generati (7 CSV + 1 JSON)

File	**Formato**	**Granularità / chiave**	**Descrizione (breve)**
users.csv	CSV	1 riga = 1 utente; PK: `user_id`	Anagrafica/attributi utente e label/target (se presenti).
exercises.csv	CSV	1 riga = 1 esercizio; PK: `exercise_id`	Catalogo esercizi (nomi, gruppi muscolari, ecc.).
workouts.csv	CSV	1 riga = 1 workout; PK: `workout_id`, FK: `user_id`	Sessioni eseguite (data, durata, volume aggregato, ecc.).
workout_sets.csv	CSV	1 riga = 1 set; PK: `set_id`, FK: `workout_id`, `exercise_id`	Dettaglio set/reps/load/RPE per ogni workout.
workout_plan.csv	CSV	1 riga = 1 elemento di piano; (chiave da specificare)	Pianificazione allenamenti (planned vs done; utile per `skip_rate`).
banister_daily.csv	CSV	1 riga = 1 giorno-utente; PK: (`user_id`, `date`)	Serie giornaliera per Banister (carico/impulso, fitness/fatigue, ecc.).

File	Formato	Granularità / chiave	Descrizione (breve)
banister_meta.csv	CSV	1 riga = 1 utente; PK: `user_id`	Parametri/meta del modello Banister (costanti, inizializzazioni, ecc.).
metadata_v2.json	JSON	1 documento; campo `version`	Metadati di generazione (seed, range date, conteggi, changelog).

.2 Dizionario delle feature

Questa appendice riporta un data dictionary (codebook) delle variabili presenti nei file generati dal generatore sintetico, includendo definizione, unità, range atteso e note di calcolo.

.2.1 `users.csv`

Tabella 2: Dizionario feature – `users.csv`

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
user_id	Identificativo univoco utente (PK), assegnato come $i + 1$.	id	≥ 1	int
start_date	Data inizio finestra utente, calcolata come `today - duration_days`.	date	ISO (YYYY-MM-DD)	string/date
end_date	Data fine finestra utente, fissata a `today`.	date	ISO (YYYY-MM-DD)	string/date
duration_days	Durata finestra (giorni), stratificata per livello: Beginner 90–270, Intermediate 240–600, Advanced 540–1080.	days	90–1080	int
weekly_freq_declared	Frequenza settimanale dichiarata (clamp e cast), con vincoli 1–6.	sess./week	1–6	int

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
split_type	Tipo split: `PPL` (p=0.7) o `FullBody` (p=0.3).	cat	{PPL, FullBody}	string
profile	Profilo comportamentale: `balanced`, `high_volume`, `high_intensity`, `inconsistent`.	cat	4 categorie	string
experience_label	Etichetta esperienza (ground-truth generatore): Beginner/Intermediate/Advanced.	cat	3 classi	string
experience_latent	Variabile latente continua (campionata uniforme).	score	0–1	float
alpha_adapt	Tasso latente di adattamento.	score	0.03–0.12	float
k_detraining	Tasso latente di detraining.	score	0.005–0.02	float
obs_noise	Rumore osservazionale latente.	score	0.5–1.5	float
resilience	Resilienza latente.	score	0.5–1.2	float
fatigue_sens	Sensibilità alla fatica latente.	score	0.6–1.4	float
rpe_report_bias	Bias individuale di reporting RPE (aggiunto all'RPE osservato).	score	-0.3–0.3	float
discipline	Disciplina (da profilo + rumore, poi clip).	score	0.1–1.0	float
motivation	Motivazione (normale, poi clip).	score	0.3–1.0	float
consistency_score	Aderenza globale: #giorni done/#giorni totali, poi clip nel range target per livello (Beginner 0.65–0.80, Intermediate 0.75–0.90, Advanced 0.85–0.95).	rate	0.65–0.95	float

.2.2 `exercises.csv`

Tabella 3: Dizionario feature – `exercises.csv`

Feature	Definizione	Range atteso	Tipo
exercise_id	Identificativo esercizio (PK) nel catalogo.	≥ 1	int
exercise_name	Nome esercizio (stringa).	–	string

Feature	Definizione	Range atteso	Tipo
target_muscle_group	Gruppo muscolare target (categoria).	categorie finite	string
split_cat	Categoria split: Push/Pull/Legs/FullBody.	categorie finite	string

.2.3 `workouts.csv`

Tabella 4: Dizionario feature – `workouts.csv`

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
workout_id	Identificativo globale workout (PK), assegnato come indice ordinato per (`user_id`, `date`).	id	≥ 1	int
user_id	Identificativo utente (FK $\rightarrow$ `users.user_id`).	id	≥ 1	int
date	Data della sessione pianificata.	date	ISO (YYYY-MM-DD)	string/date
week_index_user	Indice settimana nella carriera utente: $\lfloor (d - start)/7 \rfloor + 1$.	week	≥ 1	int
session_tag	Tag sessione in base allo split: Push/Pull/Legs o FullBody.	cat	set finito	string
workout_status	Stato: `done` oppure `skipped`.	cat	{done, skipped}	string
z_skip	Logit score del modello skip: $z = \text{logit}(p0) + w_f \cdot \log(1 + \text{fatigue}) + \epsilon$.	score	real	float
p_skip	Probabilità di skip: $\sigma(z)$ (sigmoid).	prob	0–1	float
fatigue_term	Termine fatica usato nello skip: $\log(1 + \text{fatigue})$ con cap.	score	≥ 0	float
experience_label	Etichetta esperienza associata all'utente (ridondante rispetto a `users`).	cat	3 classi	string

.2.4 workout_plan.csv

Tabella 5: Dizionario feature – workout_plan.csv

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
workout_id	Identificativo workout (FK → workouts.workout_id), propagato tramite mapping (user_id, date).	id	≥ 1	int
user_id	Identificativo utente.	id	≥ 1	int
date	Data sessione.	date	ISO (YYYY-MM-DD)	string/date
session_tag	Tag sessione (Push/Pull/Legs/Full-Body).	cat	set finito	string
exercise_id	Identificativo esercizio (FK → exercises.exercise_id).	id	≥ 1	int
sets_planned	Set pianificati per esercizio, dipendono da experience (Beg 2–4, Int 3–5, Adv 4–6) e vengono ridotti in caso di infortunio (moltiplicatore 0.6, min 1).	set	1–6	int
reps_min	Min reps pianificate (fisso).	reps	6	int
reps_max	Max reps pianificate (fisso).	reps	12	int
rest_planned_sec	Recupero pianificato tra set.	sec	{90,120,150,180}	int
rir_target	Target RIR per livello: Beg 2–4, Int 1–3, Adv 0–2.	reps	0–4	int

.2.5 workout_sets.csv

Tabella 6: Dizionario feature – `workout_sets.csv` (tabella primaria a livello set)

Feature	**Definizione / calcolo**	**Unità**	**Range atteso**	**Tipo**
set_id	Identificativo set (PK) stringa: `{user}S{counter}` (es. 0001S0000001).	id	unico	string
workout_id	Identificativo workout (FK), propagato da (`user_id`, `date`).	id	≥ 1	int
user_id	Identificativo utente.	id	≥ 1	int
date	Data sessione.	date	ISO (YYYY-MM-DD)	string/date
week_index_user	Indice settimana nella carriera utente.	week	≥ 1	int
week_type	Tipo settimana (ACWR): `normal`, `spike`, `deload`.	cat	3 categorie	string
acwr	Valore ACWR settimanale usato come moltiplicatore (normal circa 0.90–1.25; spike 1.35–1.85; deload 0.65–0.85).	ratio	≥ 0	float
session_tag	Tag sessione (Push/Pull/Legs/FullBody).	cat	set finito	string
exercise_id	Esercizio eseguito (FK).	id	≥ 1	int
set_index	Indice del set all'interno dell'esercizio nella sessione (1..sets_done).	idx	1–10	int
reps_target	Reps target campionate tra `reps_min` e `reps_max`.	reps	6–12	int
reps_done	Reps eseguite: dipendono da target e fatica, clampate (validazione: 1–50).	reps	1–50	int
load_intended_kg	Carico "inteso": q_load(intensity $\cdot$ c_max) con step 0.25 kg.	kg	≥ 0	float

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
load_done_kg	Carico eseguito: combinazione capacity-based e daily-variation, quantizzato (step 0.25 kg); missing con probabilità `p_missing_load`; clamp post-process in [2.5, 200.0].	kg	2.5–200.0 (se non missing)	float/NaN
rpe_done	RPE osservato: mix di intensità, motivazione, fatica; poi noise+bias experience-aware; quantizzato step 0.5 e clamp 1–10; missing con `p_missing_rpe`.	1–10	1–10 (se non missing)	float/NaN
rest_planned_sec	Recupero pianificato (replicato dal piano).	sec	{90,120,150,180}	int
rir_target	RIR target (replicato dal piano).	reps	0–4	int
feedback	Feedback testuale opzionale (5%), missing con `p_missing_feedback`.	text	–	string/None

.2.6 `banister_meta.csv`

Tabella 7: Dizionario feature – `banister_meta.csv`

Feature	Definizione / calcolo	Unità	Range atteso	Tipo
user_id	Identificativo utente (PK e FK).	id	≥ 1	int
tau_F	Costante di tempo Fitness per utente (campionata da normale e troncata a ≥ 7).	days	≥ 7	float
tau_D	Costante di tempo Fatigue per utente (campionata e troncata a ≥ 2).	days	≥ 2	float
beta_F	Guadagno Fitness globale (config).	coeff	> 0	float
beta_D	Guadagno Fatigue globale (config).	coeff	> 0	float

.2.7 `banister_daily.csv`

Tabella 8: Dizionario feature – `banister_daily.csv`

Feature	**Definizione / calcolo**	**Unità**	**Range atteso**	**Tipo**
user_id	Identificativo utente.	id	≥ 1	int
date	Data (giornaliera).	date	datetime	datetime
impulse	Impulso giornaliero: somma sui set del giorno di `load_done_kg` · `reps_done` · `rpe_done`/10 (con contributo 0 se load/RPE missing).	a.u.	≥ 0	float
tau_F	Parametro Banister (replicato da meta per merge).	days	≥ 7	float
tau_D	Parametro Banister (replicato da meta per merge).	days	≥ 2	float
beta_F	Parametro Banister (replicato).	coeff	> 0	float
beta_D	Parametro Banister (replicato).	coeff	> 0	float
fitness	Serie Fitness: convoluzione di `impulse` con pesi esponenziali $\exp(-i/\tau_F)$, scalata per β_F.	a.u.	real	float
fatigue	Serie Fatigue: convoluzione di `impulse` con $\exp(-i/\tau_D)$, scalata per β_D.	a.u.	real	float
TSB	Training Stress Balance: `fitness - fatigue`.	a.u.	real	float
performance	Performance scalata in [0,100] per utente: min-max scaling della serie `fitness - fatigue`.	score	0–100	float

.3 Specifica del file `metadata_v2.json`

Tabella 9: Campi del file `metadata_v2.json`

Campo	**Tipo**	**Significato**
version	string	Versione del generatore/dataset (es. “2.0”).

Campo	Tipo	Significato
date_generated	string (datetime)	Timestamp di generazione del dataset.
seed	integer	Seed random per riproducibilità.
n_users	integer	Numero utenti generati.
n_workouts	integer	Numero workouts totali nel dataset.
n_sets	integer	Numero set totali (righe attese in `workout_sets.csv`).
experience_distribution	object	Conteggio utenti per livello (Beginner/Intermediate/Advanced).
skip_rates	object	Skip rate medio per livello.
date_range.min/max	string (datetime)	Estremi temporali del dataset.
changelog	array[string]	Elenco modifiche e razionali (con riferimenti bibliografici nel testo).