Università
Ca'Foscari
Venezia

Corso di Laurea
in Informatica

Tesi di Laurea

Sviluppo di una metrica composita per la valutazione dei modelli di previsione con serie temporali

Relatrice
Ch.ma Prof.ssa Ilaria Prosdocimi

Laureando
Alessandro Conte
Matricola: 877148

Anno Accademico
2023/2024

Sommario

Questa tesi tratta lo sviluppo di una metrica composita in grado di combinare in maniera equilibrata le metriche di errore più comunemente utilizzate nell'analisi delle serie temporali (RMSE, MAE, MASE, MAPE), sfruttandone i vari punti di forza e cercando di far fronte alle loro limitazioni, come la sensibilità agli outliers e la difficoltà di comparazione tra metriche. Il progetto è stato sviluppato durante il periodo di tirocinio formativo presso l'azienda "Humco" e si concentra sulla progettazione, implementazione e validazione della metrica composita su dataset reali, utilizzando diversi modelli previsionali (ARIMA, SARIMA, Holt-Winters, Prophet). I risultati dimostrano come la metrica sia in grado di fornire un valore rappresentativo dell'errore complessivo di un modello, permettendo quindi di classificare in maniera più efficiente i modelli in base alla loro bontà predittiva. Tuttavia, la metrica composita risulta essere poco precisa in alcuni scenari, pertanto al termine di questa tesi, vengono discussi i possibili sviluppi futuri del progetto volti a migliorarne l'affidabilità. Con un maggior numero di scenari analizzati e ulteriori miglioramenti la metrica potrebbe rappresentare una componente utile nel mondo dell'analisi delle serie temporali, essendo uno strumento semplice ed efficace per valutare le prestazioni dei modelli predittivi.

Indice

1 Introduzione

1.1 Contesto e motivazione

Questa tesi nasce all'interno di un tirocinio svolto presso l'azienda Humco, con l'obiettivo di sviluppare una tecnica per rendere più efficiente l'analisi dei risultati di alcuni modelli previsionali in termini di bontà predittiva. La valutazione delle previsioni generate dai modelli su serie storiche è un aspetto molto importante al giorno d'oggi, soprattutto in settori come quello economico, energetico e della sanità. Un valutazione accurata consente di ottimizzare strategie decisionali e ridurre rischi associati all'incertezza dei dati, ed avviene attraverso metriche che quantificano l'errore tra valori previsti e quelli reali. Tuttavia, ogni metrica presenta delle limitazioni, approfondite nei capitoli successivi, che possono essere ridotte combinando in modo equilibrato le metriche tra loro. Grazie alla metrica composita è possibile combinare le varie metriche d'errore in un unico valore che sfrutta i vari punti di forza di ogni metrica cercando di limitarne le debolezze. La metrica proposta può essere impiegata per confrontare modelli previsionali in scenari reali, migliorando la selezione del modello più adatto a seconda delle caratteristiche dei dati analizzati. L'esigenza di una metrica composita è emersa direttamente dall'attività di ricerca e sviluppo condotta in Humco, dove la valutazione accurata delle previsioni su dati aziendali è fondamentale per ottimizzare i processi decisionali.

1.2 Obiettivi della tesi

L'obiettivo di questa tesi è quello di definire una metrica di bontà degli algoritmi di previsione delle serie storiche che riassuma quattro metriche di errore già esistenti: RMSE, MAE, MAPE e MASE. É importante includere un'analisi del comportamento della metrica in funzione al tipo di dato analizzato, confrontandola con le metriche tradizionali per valutarne l'efficacia. Infine è necessario identificare i limiti dello studio e gli eventuali miglioramenti del progetto che potranno essere sviluppati in futuro.

1.3 Struttura della Tesi

La tesi è composta da sei capitoli:

- **Capitolo 1 : Introduzione.** In questo capitolo viene fatta un'introduzione al contesto e alla motivazione della tesi, vengono definiti gli obiettivi e la struttura di quest'ultima.

- **Capitolo 2 : Fondamenti teorici.** Questo capitolo introduce al lettore tutte le nozioni teoriche necessarie per apprendere lo sviluppo del progetto. Sono incluse le definizioni di serie temporali, dei vari modelli di previsione utilizzati e delle metriche di errore trattate nel corso della tesi.

- **Capitolo 3 : Progettazione della metrica composita.** Qui si introducono i limiti delle metriche singole e si definiscono i requisiti della metrica composita. Successivamente viene spiegato da quali componenti è formata e come è possibile calcolarla.

- **Capitolo 4 : Implementazione e sperimentazione.** Questo capitolo illustra l'implementazione della metrica allenando i modelli con dataset reali. Si fornisce una descrizione dei set di dati utilizzati, dei parametri scelti per i vari modelli e delle tecniche di preprocessamento dei dati.

- **Capitolo 5 : Risultati e valutazione.** In questo capitolo vengono commentati i risultati ottenuti dall'analisi dei dataset, confrontando le metriche singole con la metrica composita dimostrandone la correttezza. Infine vengono affrontati i vantaggi e le limitazioni che l'utilizzo della metrica composita comporta.

- **Capitolo 6 : Conclusioni.** Per concludere vengono descritti i risultati ottenuti tramite questa tesi ed i limiti dello studio, seguiti dai possibili sviluppi futuri.

2 Fondamenti teorici

In questo capitolo vengono introdotte le nozioni teoriche necessarie al fine di comprendere lo sviluppo del progetto. Sono presenti le definizioni di serie temporali, dei modelli utilizzati, delle metriche e delle metodologie applicate in questo progetto.

2.1 Serie temporali

Le serie temporali (o storiche) sono delle sequenze di osservazioni ordinate cronologicamente, ottenute tramite misurazioni ripetute ad intervalli di tempo sia regolari che irregolari. Vengono rappresentate mediante grafici, nei quali una delle due assi è sempre espressa in funzione del tempo. L'analisi delle serie temporali è una tipologia di analisi tramite la quale è possibile osservare come le variabili cambiano nel tempo, con lo scopo di effettuare delle previsioni sull'andamento futuro dei dati. Un esempio tipico può essere la previsione della domanda di energia elettrica da parte di un'azienda fornitrice, che vuole ottimizzare la produzione e ridurre gli sprechi. Questo tipo di analisi solitamente richiede un gran numero di dati per garantire coerenza e affidabilità. Un set di dati esteso garantisce di avere una dimensione del campione rappresentativa e che l'analisi possa escludere i dati rumorosi. Garantisce inoltre che eventuali tendenze o modelli scoperti non siano valori anomali e possano tenere conto della varianza stagionale. Un altro punto importante è che l'analisi delle serie temporali viene effettuata quando si lavora con dati non stazionari, quindi dati che fluttuano costantemente nel tempo o che possono dipendere da questo. Infatti settori come ad esempio quello finanziario o di vendita utilizzano molto questo tipo di analisi, dato che la moneta ed i prezzi variano costantemente.

2.1.1 Classificazione delle serie temporali

Le serie temporali possono essere suddivise in due categorie principali: univariate e multivariate. Le prime sono caratterizzate dalla fluttuazione di una sola variabile nel tempo, le seconde invece sono caratterizzate dall'oscillazione di più variabili. Per fare un esempio, i dati raccolti da un sensore in grado di misurare ogni secondo la temperatura di una stanza possono essere rappresentati tramite serie univariate, mentre la raccolta dati effettuata da una stazione meteorologica, la quale ogni ora raccoglie dati di temperatura, umidità, pressione atmosferica e velocità del vento in una determinata località, tramite serie multivariate.

2.1.2 Caratteristiche delle serie temporali

Le serie temporali presentano alcune caratteristiche di cui bisogna tenere conto. Una caratteristica importante è la stagionalità. Questa si manifesta quando le fluttua-

zioni dei dati avvengono in maniera regolare e prevedibile ad intervalli di tempo definiti come giorni, mesi o anni. Un esempio classico di stagionalità è l'analisi del consumo energetico, che tende ad aumentare nei mesi invernali e estivi.
A differenza della stagionalità, la ciclicità è una caratteristica delle serie temporali che si riferisce a cambiamenti in periodi di tempo indefiniti. Ad esempio, i cicli economici possono avere una durata pluriennale e non seguire una periodicità fissa, rendendoli più difficili da prevedere rispetto alla stagionalità.
Un'altra caratteristica rilevante da analizzare quando si parla di serie temporali è la tendenza (o trend). Essa viene osservata quando il pattern nei dati mostra un movimento della serie verso valori più alti o più bassi in un lungo periodo di tempo, come ad esempio l'andamento del PIL di un paese, che può mostrare una crescita costante nel tempo o una decrescita in periodi di recessione economica.

2.1.3 Stagionalità nelle serie temporali

La stagionalità è la componente delle serie temporali che rappresenta la ripetizione regolare di schemi o pattern a intervalli di tempo fissi. Questa caratteristica è spesso legata a fenomeni ciclici, come le variazioni stagionali nelle vendite di un determinato prodotto, nella domanda di energia elettrica o nei flussi di traffico. La sua identificazione e analisi sono importanti per migliorare la qualità delle previsioni e comprendere la dinamica sottostante dei dati. Esistono diversi approcci per individuare la stagionalità in una serie temporale, nel contesto di questo progetto sono state utilizzate due tecniche: la decomposizione STL(Seasonal-Trend decomposition using Loess) e la FFT (Fast Fourier Transform) che verranno descritte nelle sezioni successive.

2.1.4 Ciclicità nelle serie temporali

La ciclicità nelle serie storiche si riferisce a variazioni irregolari dei dati che non rispettano un periodo preciso di tempo. Al contrario della stagionalità, la fluttuazione dei dati non è strettamente legata a fenomeni stagionali ma a dinamiche a lungo termine, come nel caso dell'analisi delle vendite di automobili che potrebbe mostrare periodi di crescita seguiti da periodi di calo, riflettendo un ciclo economico.

2.1.5 Tendenza delle serie temporali

La tendenza, o trend, è la componente che permette di osservare l'andamento delle serie storiche in un certo periodo di tempo. In base alle caratteristiche del fenomeno osservato, è possibile riconoscere tre tipi di tendenze: crescente, decrescente e stazionaria.

Tendenza crescente

La tendenza crescente si verifica quando l'analisi delle serie temporale mostra un andamento generale dei dati che aumenta con il passare del tempo

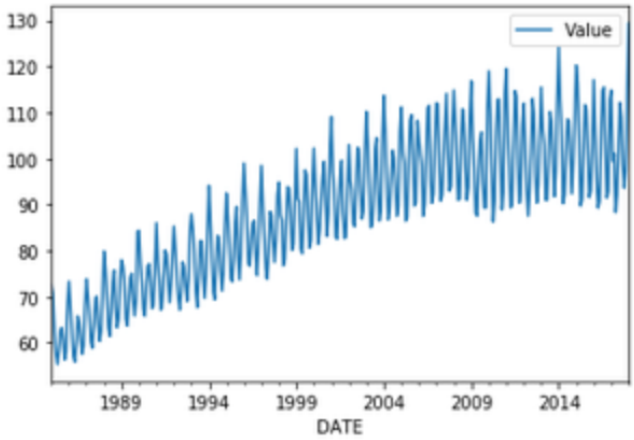

Figura 2.1: Esempio di serie temporali con tendenza crescente e stagionalità (Fonte : Tejalk 2024)

Tendenza decrescente

La tendenza decrescente si verifica quando l'analisi delle serie temporale mostra un andamento generale dei dati che diminuisce con il passare del tempo

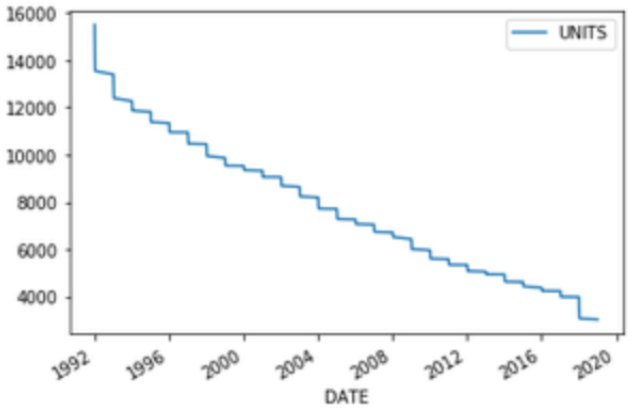

Figura 2.2: Esempio di serie temporali con tendenza decrescente senza stagionalità (Fonte : Tejalk 2024)

Tendenza stazionaria (Assenza di trend)

La tendenza stazionaria si verifica quando tramite l'analisi della serie temporale non si osserva alcun pattern nei dati

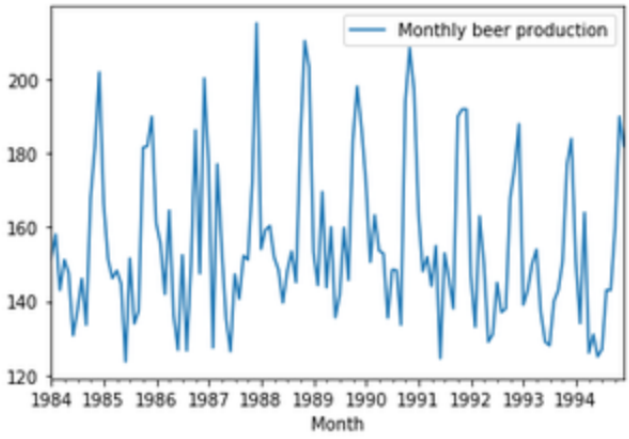

Figura 2.3: Esempio di serie temporali con assenza di trend (Fonte : Tejalk 2024)

Linearità nelle serie temporali

Indipendentemente dal tipo, una serie temporale può essere lineare o non lineare:

- Lineare: l'incremento o decremento della tendenza è costante nel tempo, ed è rappresentabile con un modello del tipo:

$$T_t = \alpha + \beta t$$

- Non Lineare: l'incremento o decremento della tendenza non è costante nel tempo ma segue un comportamento più complesso, come ad esempio una curva esponenziale o polinomiale.

Identificare la tendenza è un processo importante per comprendere il comportamento dei dati sul lungo termine, ed è spesso necessario separarla dalle componenti stagionali e cicliche.[1]

2.1.6 Relazioni tra stagionalità, ciclicità e tendenza

Queste tre componenti possono essere presenti all'interno di un'unica serie temporale ed è opportuno analizzarle separatamente per ottenere un modello predittivo accurato. Ad esempio, analizzando una serie temporale riguardante le vendite di un prodotto, è possibile osservare:

- Una tendenza crescente causata da una maggiore domanda del prodotto
- Cicli economici legati alla stagionalità della spesa
- Fluttuazioni stagionali annuali provocate da periodi in cui la richiesta aumenta (come durante le festività)

La decomposizione delle serie temporali in questi tre elementi è un passaggio cruciale nell'analisi, e viene generalmente effettuata tramite metodi come la decomposizione classica (additiva o moltiplicativa) oppure tramite tecniche più avanzate come la decomposizione STL (*Seasonal and Trend decomposition using Loess*).

[1] Galdino 2023.

2.2 Metodo di decomposizione STL

Il metodo STL, che sta per *Seasonal and Trend decomposition using* LOESS (*Locally Estimated Scatterplot Smoothing*, ovvero un metodo per stimare relazioni non lineari) è utile per scomporre una serie temporale in tre componenti: trend, stagionalità e residui. Grazie a questa scomposizione diventa più facile scoprire pattern nascosti nei dati e ottimizzare le strategie utilizzate. Viene utilizzato in questo progetto per calcolare una componente importante per la ponderazione delle varie metriche.
La decomposizione *STL* esprime una serie temporale come segue:

$$y_t = S_t + T_t + R_t$$

Dove:

- y_t è il valore della serie temporale al tempo t
- S_t è il valore della componente stagionale al tempo t
- T_t è il valore della componente di tendenza al tempo t
- R_t è il valore della componente restante (o rumore) al tempo t

2.2.1 Procedura della decomposizione STL

STL utilizza un approccio iterativo per separare le tre componenti. Il metodo può essere suddiviso in tre fasi:

1. Estrazione della componente di tendenza

La componente di tendenza rappresenta la direzione che prendono i dati sul lungo periodo. Per estrarre questa componente, *STL* applica un processo di smoothing:

- LOESS : *Locally Estimated Scatterplot Smoothing (Loess)* è una tecnica utilizzata per adattare una curva regolare ai punti dati. Grazie ad essa è possibile catturare meglio la direzione complessiva dei dati, calcolando la media delle fluttuazioni a breve termine.
- *Iterative Refinement* : STL applica iterativamente lo *smoothing* Loess per rendere più accurata la componente di tendenza. Utilizzando un'ampia finestra di *smoothing*, il processo filtra le variazioni ad alta frequenza, lasciando una rappresentazione fluida della progressione a lungo termine

2. Estrazione della stagionalità

La stagionalità cattura i pattern periodici che si ripetono in un certo intervallo di tempo che può essere giornaliero, mensile o annuale. Per isolare la componente stagionale, STL esegue le seguenti operazioni:

- *Detrending* : Per prima cosa, la componente trend viene sottratta dalla serie temporale originale, ottenendo così una serie detrended che contiene principalmente componenti stagionali e di rumore.

- Suddivisione in sottoserie stagionali : La serie *detrended* viene successivamente divisa in sottoserie stagionali. Ad esempio, per i dati mensili, ogni sottoserie contiene i *data points* corrispondenti allo stesso mese in anni differenti.

- *Smoothing* LOESS su sottoserie : Lo *smoothing* LOESS viene applicato ad ogni sottoserie separatamente per catturare il pattern ripetuto all'interno di ogni stagione. Questo passaggio assicura che la componente stagionale spieghi accuratamente le variazioni periodiche

3. Estrazione del rumore

Il rumore, o componente rimanente, rappresenta le fluttuazioni casuali o irregolari presenti nei dati che non sono spiegate dalla componente di tendenza o stagionale. Il rumore viene calcolato tramite:

- Calcolo residuo : dopo l'estrazione di trend e stagionalità, la componente rumore è semplicemente il residuo rimasto dalla serie originale. Questo viene calcolato sottraendo sia i componenti trend che stagionali dai dati originali.

- $R_t = y_t - T_t - S_t$: Matematicamente, il rumore (R_t) è la differenza tra la serie temporale originale (y_t) e la somma delle componenti di tendenza (T_t) e stagionale (S_t).

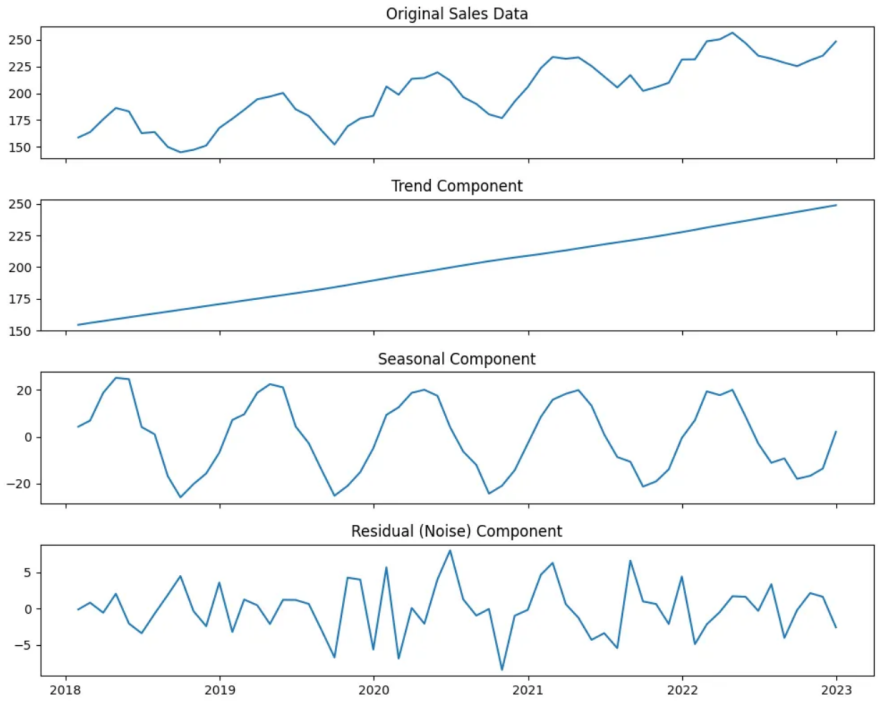

Figura 2.4: Visualizzazione di risultati ottenuti tramite metodo STL (Fonte: Kis 2024)

L'utilizzo di STL presenta diversi vantaggi rispetto alla decomposizione classica e ad altri metodi :

- Gestisce qualsiasi tipo di stagionalità, non solo quella mensile e trimestrale.

- La componente stagionale può variare nel tempo con una velocità controllata dall'utente.
- A discrezione dell'utente, STL può essere un metodo robusto rispetto a valori anomali, in modo che la presenza di osservazioni anomale occasionali non influenzi la stima dei componenti di trend e stagionalità.

2.3 Fast Fourier Transform (FFT)

La *Fast Fourier Transform* (FFT) è un algoritmo che calcola in modo efficiente la Trasformata di Fourier Discreta (DFT) e la sua inversa. La DFT converte una serie temporale dal dominio del tempo al dominio delle frequenze, decomponendola in una somma di sinusoidi con diverse ampiezze, frequenze e fasi.
Matematicamente, la DFT è definita come:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}, k = 0, 1, ..., N-1$$

dove:

- x_n sono i valori della serie temporale originale
- X_k sono i coefficienti della trasformata (cioè l'ampiezza di ciascuna frequenza)
- N è la lunghezza della serie[2]

L'algoritmo FFT è stato utilizzato durante lo svolgimento di questo progetto per trovare i possibili periodi dominanti delle serie temporali analizzate, da inserire come parametro di modelli SARIMA e Holt-Winters.

2.4 Modelli di previsione

Un modello di previsione è uno strumento che consente di stimare valori futuri di una serie temporale a partire dai dati storici. In questa sezione veranno presentati i modelli di previsione utilizzati per confrontare le varie metriche di errore con la metrica composita sviluppata, evidenziando le loro caratteristiche principali e il loro comportamento sulle serie analizzate.

2.4.1 ARIMA

ARIMA (*Autoregressive Integrated Moving Average*) è un modello di analisi che utilizza serie temporali per comprendere al meglio il dataset o per effettuare predizioni future. É un modello autoregressivo, cioè un modello che effettua previsioni basandosi su dati passati. Il fatto che sia integrato (I) significa che rappresenta la differenziazione delle osservazioni grezze per consentire alla serie temporale di diventare stazionaria. Inoltre utilizza la media mobile (MA) cioè incorpora la dipendenza tra un'osservazione e un errore residuo da un modello di media mobile applicato alle osservazioni ritardate[3]. Il modello ARIMA utilizza tre parametri:

[2]Maklin 2024.
[3]Hayes 2024.

- p : indica il numero di lag (ritardi) della serie che vengono utilizzati per prevedere il valore attuale. Indica l'ordine del termine autoregressivo
- d : indica il numero di differenziazioni necessarie per rendere la serie stazionaria. Indica il grado di differenziazione
- q : indica il numero di termini della media mobile utilizzati per correggere errori nei residui

Tre parametri definiscono un modello ARIMA non stagionale: l'ordine dell'espressione autoregressiva p, il grado di differenziazione d, cioè il numero di differenziazioni necessarie per rendere la serie stazionaria, e infine l'ordine dell'espressione della media mobile q. Più nello specifico, il parametro p si riferisce al numero di occorrenze y che verranno utilizzate come indicatori per prevedere il valore attuale e il parametro q si riferisce al numero di errori previsionali non abbastanza utili da essere inclusi nel modello ARIMA.[4]
L'equazione generale del modello ARIMA è la seguente:

$$y_t = \mu + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q}$$

Dove:

- y_t è il valore osservato della serie temporale al tempo t.
- μ è la costante (opzionale).
- ϕ_i sono i coefficienti autoregressivi (AR) associati ai lag della serie temporale.
- y_{t-i} rappresenta i valori passati della serie temporale.
- θ_j sono i coefficienti della media mobile (MA) associati ai lag degli errori.
- ε_{t-j} rappresenta gli errori passati (shock aleatori).
- p è l'ordine della componente autoregressiva (AR).
- q è l'ordine della componente di media mobile (MA).[5]

2.4.2 SARIMA

Il modello SARIMA (*Seasonal Autoregressive and Integrated Moving Average*) estende il modello ARIMA aggiungendo dei parametri stagionali.

- p, d, q : Questi parametri hanno lo stesso significato di quelli presenti nel modello ARIMA(p,d,q), rappresentando rispettivamente l'autoregressività, la differenziazione e la componente di media mobile.
- P : Rappresenta l'ordine del processo autoregressivo (AR) stagionale, indicando il valore dei lag nella componente stagionale

[4] Ersoz et al. 2022.
[5] Ersoz et al. 2022.

- D : Questo parametro indica l'ordine di integrazione stagionale, simile a d ma applicato alla componente stagionale.
- Q : Questo parametro indica l'ordine del processo di media mobile (MA) stagionale, che cattura i termini di errore di lag nella componente stagionale.
- m : Questo parametro rappresenta la frequenza, indicando il numero di osservazioni per ciclo stagionale. La lunghezza del ciclo dipende dal set di dati e dalla natura della stagionalità presente.

Il modello SARIMA$(p,d,q)(P,D,Q)m$ consente una modellazione completa delle componenti non stagionali e stagionali nei dati delle serie temporali, fornendo un quadro potente per la previsione e l'analisi.[6]
La formula generale del modello SARIMA è la seguente:

$$\Phi_P(B^s)\phi_p(B)(1-B)^d(1-B^s)^D y_t = \Theta_Q(B^s)\theta_q(B)\varepsilon_t$$

Dove:

- B è l'operatore di ritardo, tale che $By_t = y_{t-1}$.
- $(1-B)^d$ rappresenta la differenziazione non stagionale di ordine d.
- $(1-B^s)^D$ rappresenta la differenziazione stagionale di ordine D con periodo stagionale s.
- $\phi_p(B) = 1 - \phi_1 B - \cdots - \phi_p B^p$ è il polinomio autoregressivo di ordine p.
- $\theta_q(B) = 1 - \theta_1 B - \cdots - \theta_q B^q$ è il polinomio della media mobile di ordine q.
- $\Phi_P(B^s) = 1 - \Phi_1 B^s - \cdots - \Phi_P B^{Ps}$ è il polinomio autoregressivo stagionale di ordine P.
- $\Theta_Q(B^s) = 1 - \Theta_1 B^s - \cdots - \Theta_Q B^{Qs}$ è il polinomio della media mobile stagionale di ordine Q.
- ε_t è un errore bianco con media zero e varianza costante.[7]

2.4.3 Holt-Winters

Il modello Holt-Winters è utilizzato per la previsione delle serie temporali, basata sullo smoothing esponenziale dei dati. Questo approccio assegna pesi esponenzialmente decrescenti ai valori della serie nel tempo, dando maggiore importanza ai dati più recenti rispetto a quelli passati. Il metodo di Holt-Winters si distingue per la capacità di gestire tre componenti fondamentali della serie temporale: livello, trend e stagionalità.
La formulazione generale del modello può essere espressa in due modi differenti in base al tipo di metodo utilizzato: addittivo o moltiplicativo.

[6] Kwarteng e Andreevich 2024.
[7] Alonso et al. 2021.

Metodo additivo

Il tipo di modello additivo è adatto quando la componente stagionale è stabile nel tempo e non dipende dal livello generale della serie. Possiamo esprimerlo tramite la seguente formula:

$$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$$

$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$$

$$s_t = \gamma(y_t - \ell_t) + (1-\gamma)s_{t-m}$$

$$\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m(k+1)}$$

Dove:

- ℓ_t è il livello della serie al tempo t;
- b_t è il trend al tempo t;
- s_t è la componente stagionale al tempo t;
- α è il parametro di smoothing per il livello, con $0 < \alpha < 1$;
- β^* è il parametro di smoothing per il trend, con $0 < \beta^* < 1$;
- γ è il parametro di smoothing per la stagionalità, con $0 < \gamma < 1$;
- m è la periodicità della stagionalità (ad esempio, $m = 12$ per dati mensili);
- $\hat{y}_{t+h|t}$ è la previsione per h passi avanti al tempo t;
- k è il numero di cicli stagionali completati tra t e $t+h$, calcolato come $k = \lfloor (h-1)/m \rfloor$.[8]

Metodo moltiplicativo

Il metodo moltiplicativo è più appropriato quando la stagionalità varia proporzionalmente al livello della serie, risultando più adatto in presenza di trend esponenziali. Viene espresso tramite la seguente formula matematica:

$$\ell_t = \alpha\left(\frac{y_t}{s_{t-m}}\right) + (1-\alpha)(\ell_{t-1} + b_{t-1})$$

$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$$

$$s_t = \gamma\left(\frac{y_t}{\ell_t}\right) + (1-\gamma)s_{t-m}$$

$$\hat{y}_{t+h|t} = (\ell_t + hb_t)s_{t+h-m(k+1)}$$

Dove:

[8] R. J. Hyndman e Athanasopoulos 2018.

- ℓ_t è il livello della serie al tempo t;
- b_t è il trend al tempo t;
- s_t è la componente stagionale al tempo t;
- α è il parametro di smoothing per il livello, con $0 < \alpha < 1$;
- β^* è il parametro di smoothing per il trend, con $0 < \beta^* < 1$;
- γ è il parametro di smoothing per la stagionalità, con $0 < \gamma < 1$;
- m è la periodicità della stagionalità (ad esempio, $m = 12$ per dati mensili);
- $\hat{y}_{t+h|t}$ è la previsione per h passi avanti al tempo t;
- k è il numero di cicli stagionali completati tra t e $t + h$, calcolato come $k = \lfloor (h-1)/m \rfloor$.[9]

Questa metodologia rappresenta una soluzione efficace per la previsione di serie temporali con stagionalità, garantendo un buon compromesso tra semplicità computazionale e accuratezza delle previsioni.[10]

2.4.4 Prophet

Prophet è un modello predittivo sviluppato da Facebook che incorpora parametri per trend e stagionalità in modo da aiutare a modellare la previsione e fornire una performance migliore con serie temporali che contengono stagionalità. Grazie alla sua struttura parametrica flessibile, il modello è in grado di catturare le variazioni a lungo termine e i pattern stagionali, migliorando l'accuratezza delle previsioni.
Il modello Prophet può essere espresso tramite la seguente formula:

$$y_t = g_t + s_t + h_t + \epsilon_t$$

Dove:

- y_t è il valore osservato della serie temporale al tempo t.
- g_t rappresenta il trend, ovvero la crescita a lungo termine della serie.
- s_t rappresenta la stagionalità, che cattura le variazioni periodiche ricorrenti.
- h_t rappresenta gli eventi speciali o effetti delle festività, se presenti.
- ϵ_t è il termine di errore o rumore, che rappresenta la componente non spiegata dal modello.[11]

[9] R. J. Hyndman e Athanasopoulos 2018.
[10] R. J. Hyndman e Athanasopoulos 2018.
[11] Ersoz et al. 2022.

2.5 Metriche di valutazione delle previsioni

Le metriche di errore nella previsione delle serie temporali sono essenziali per una corretta valutazione di accuratezza di un modello. Di seguito verranno presentate le varie metriche utilizzate durante lo sviluppo di questo progetto, esse risultano essere complementari: nessuna è da considerarsi migliore delle altre, la scelta dipende dal contesto dell'analisi e dalle esigenze del modello predittivo.

2.5.1 RMSE

L'RMSE (*Root Mean Squared Error*) è una misura statistica utilizzata per valutare le prestazioni dei modelli predittivi. Rappresenta la radice quadrata della media degli errori, tra i valori previsti e quelli osservati, elevati al quadrato, riuscendo a fornire un'indicazione della dispersione degli errori. É possibile calcolare questa metrica utilizzando la seguente formula matematica:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$

Dove:

- y_t è il valore osservato al tempo t.
- $\hat{y}_t$ è il valore predetto al tempo t.
- n è il numero totale di osservazioni.
- La differenza $(y_t - \hat{y}_t)$ rappresenta l'errore di previsione.
- L'errore viene elevato al quadrato, sommato e poi diviso per n, prima di essere radicato per ottenere l'RMSE.[12]

Utilizzare la radice quadrata del *MSE* non altera l'effettivo ranking dei modelli ma produce una metrica con la stessa unità di y, la quale rappresenta lo "standard" di errore per errori normalmente distribuiti. Questa metrica penalizza maggiormente gli errori più grandi, rendendola utile in contesti in cui questo tipo di errori deve essere evitato.

2.5.2 MAE

L'errore assoluto medio o MAE (*Mean Absolute Error*) è una metrica che calcola l'entità media degli errori assoluti tra i valori previsti e quelli effettivi.[13] Può essere calcolata tramite la seguente formula:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$$

Dove:

[12] Hodson 2022.
[13] Xia, Bi e C. Wang 2023.

- y_t è il valore osservato al tempo t.
- $\hat{y}_t$ è il valore predetto al tempo t.
- n è il numero totale di osservazioni.
- La differenza $(y_t - \hat{y}_t)$ rappresenta l'errore di previsione.[14]

Questa metrica è facile da interpretare dato che anch'essa è espressa nella stessa unità della variabile osservata e risulta essere meno sensibile agli outliers rispetto all'RMSE.

2.5.3 MAPE

Il MAPE (*Mean Absolute Percentage Error*) è una metrica che indica l'errore medio assoluto percentuale, rendendo più facile il confronto tra serie con scale diverse. La formula per calcolarlo è la seguente:

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{y_t - \hat{y}_t}{y_t} \right|$$

Dove:

- y_t è il valore osservato al tempo t.
- $\hat{y}_t$ è il valore predetto al tempo t.
- n è il numero totale di osservazioni.
- La differenza $(y_t - \hat{y}_t)$ rappresenta l'errore di previsione.
- L'errore viene diviso per il valore osservato y_t, trasformandolo in una percentuale d'errore relativa.
- Si prende il valore assoluto dell'errore percentuale e si calcola la media moltiplicata per 100 per ottenere il MAPE.

Dato che il MAPE è una misura percentuale, il confronto tra diverse serie temporali è facilitato se essa viene utilizzata. Tuttavia, questa metrica penalizza maggiormente gli errori su valori bassi rispetto a quelli su valori alti e inoltre, in presenza di valori nulli, questa non risulta utilizzabile a causa della divisione per zero.

2.5.4 MASE

Il MASE (*Mean Absolute Scaled Error*) confronta l'errore assoluto medio di un modello con quello di un modello naïve basato sul ritardo stagionale :

$$\text{MASE} = \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{\frac{1}{n-m} \sum_{t=m+1}^{n} |y_t - y_{t-m}|}$$

Dove:

[14] Hodson 2022.

- y_t è il valore osservato al tempo t.
- $\hat{y}_t$ è il valore predetto al tempo t.
- n è il numero totale di osservazioni.
- m è il periodo della stagionalità (se presente), altrimenti $m = 1$.
- Il numeratore rappresenta la somma degli errori assoluti delle previsioni.
- Il denominatore è la media degli errori assoluti di un modello di riferimento naïve basato sul ritardo m.
- Il rapporto tra i due valori fornisce il MASE, che misura la performance rispetto al modello naïve.

É una metrica normalizzata quindi, indipendentemente dalla scala dei dati, permette confronti più equi tra serie diverse. Risulta essere utile per serie stagionali grazie al confronto con il modello naïve ma richiede un valore di m appropriato per essere significativa.

2.6 Normalizzazione

Nel contesto delle serie temporali e della valutazione delle prestazioni dei modelli predittivi, la normalizzazione è un processo fondamentale per rendere confrontabili tra loro metriche con scale diverse. Questo è particolarmente utile quando si lavora con metriche di errore che hanno unità di misura differenti o range di valori molto diversi tra loro. All'interno di questa tesi verranno menzionate tre tecniche di normalizzazione: min-max, min-max versione robusta e sigmoidale.

2.6.1 Normalizzazione Min-Max

La normalizzazione min-max è una tecnica che ridimensiona i valori di una variabile in un intervallo compreso tra un minimo e un massimo prefissato, solitamente tra 0 e 1. Questo metodo preserva la distribuzione originale dei dati ma è sensibile agli outlier, poiché il valore massimo e minimo determinano direttamente la trasformazione. La formula per la normalizzazione min-max è la seguente:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

dove:

- X è il valore originale,
- $X_{\min}$ e $X_{\max}$ sono, rispettivamente, il valore minimo e massimo del dataset,
- X' è il valore normalizzato.

Questa trasformazione garantisce che i valori trasformati rientrino nell'intervallo [0,1], mantenendo le proporzioni tra i dati. Tuttavia, la presenza di valori estremi può distorcere la distribuzione dei dati normalizzati.

2.6.2 Normalizzazione Min-Max robusta

La normalizzazione min-max robusta è una variante della tecnica vista in precedenza, che riduce l'impatto degli outlier. Invece di utilizzare il minimo e il massimo assoluti del dataset, questa tecnica sfrutta i quantili della distribuzione dei dati, in particolare il primo quartile ($Q1$) e il terzo quartile ($Q3$). La formula è la seguente:

$$X' = \frac{X - Q1}{Q3 - Q1}$$

dove:

- $Q1$ è il primo quartile (25° percentile),
- $Q3$ è il terzo quartile (75° percentile),
- X' è il valore normalizzato.

L'uso dell'intervallo interquartile ($Q3 - Q1$) permette di ridurre l'influenza dei valori estremi e rende la normalizzazione più robusta rispetto alla versione standard. Questa tecnica è particolarmente utile quando i dati contengono outlier, poiché non vengono considerati i valori più estremi del dataset.

2.6.3 Normalizzazione Sigmoidale

La normalizzazione sigmoidale utilizza la funzione sigmoide per trasformare i valori in un intervallo compreso tra 0 e 1, introducendo un effetto di compressione che enfatizza le differenze tra i valori centrali e riduce l'impatto dei valori più estremi. La formula della trasformazione sigmoide è:

$$X' = \frac{1}{1 + e^{-X}}$$

dove:

- e è il numero di Nepero,
- X è il valore originale,
- X' è il valore normalizzato.

Questa tecnica è utile per ridurre l'influenza dei valori anomali senza eliminarli, rendendo i dati più gestibili in modelli che richiedono input con range ben definiti. Tuttavia, rispetto alla normalizzazione min-max, la trasformazione sigmoide può ridurre le differenze tra valori di grandezza molto diversa, appiattendo le variazioni nei dati.

3 Progettazione della metrica composita

L'analisi tradizionale delle prestazioni dei modelli di previsione si basa su metriche di errore standard come RMSE, MAE, MAPE, MASE. Ogni metrica però ha i suoi limiti, che possono portare ad avere degli scarsi risultati in termini di valutazione delle performance dei modelli. Per questo motivo nasce l'esigenza di sviluppare una metrica composita che riesce a combinare i vantaggi delle singole metriche di errore riducendone allo stesso tempo gli svantaggi. Questo capitolo contiene tutte le informazioni relative alla progettazione della metrica, i requisiti necessari, le scelte effettuate ed il calcolo della metrica vera e propria.

3.1 Limiti delle metriche singole e necessità di una metrica composita

L'idea di sviluppo della metrica composita nasce dal fatto che il tipo di analisi basato su metriche di errore come RMSE, MAE, MAPE, MASE non fornisce una valutazione completa delle prestazioni in tutti i contesti. Infatti analizzando le singole metriche ci si rende conto dei limiti che ognuna di queste ha, il RMSE ad esempio, è vantaggioso in contesti in cui gli errori grandi devono essere penalizzati e inoltre, come anche il MAE, non è normalizzato e dipende dalla scala dei dati che si sta analizzando. Il MAPE non è definito in situazioni in cui sono presenti valori nulli ed ha la tendenza di favorire modelli che sottostimano i valori reali ed infine il MASE permette confronti migliori tra i modelli ma la sua interpretazione non è immediata.
L'utilizzo della metrica composita consente:

- Bilanciamento degli errori : dato che le varie metriche sono combinate tra loro, la valutazione risulta essere più equa grazie all'attenuazione degli errori che ogni singola metrica possiede.
- Adattabilità al dataset : grazie all'introduzione della ponderazione delle metriche in base alla stagionalità e trend dei dati, la metrica risulta adattarsi meglio alla situazione in cui si lavora.
- Efficienza nel confronto : l'utilizzo di una metrica unica consente di comparare in modo efficiente le prestazioni dei modelli su dataset differenti, senza il bisogno di analizzare singolarmente quattro metriche di errore distinte.
- Flessibilità : i pesi delle metriche possono essere modificati in modo da favorire determinati aspetti in contesti specifici, e questo garantisce una maggiore robustezza in situazioni diverse.

Nonostante i vantaggi nell'utilizzo della metrica composita è importante ponderare le metriche con criterio, evitando di assegnare pesi troppo arbitrari. Un assegnazione di pesi errata potrebbe ridurre l'efficacia della metrica ed è quindi opportuno basarsi su criteri come ad esempio la forza della stagionalità e del trend che discuteremo nelle prossime sezioni.

3.2 Definizione dei requisiti per una metrica composita

Una metrica composita è ben costruita se vengono identificati correttamente i criteri fondamentali che ne garantiscono l'efficacia per quanto riguarda la valutazione dei modelli di previsione. É necessario stabilire i requisiti che assicurano un utilizzo coerente e significativo di ogni metrica, dato che ognuna di queste fornisce informazioni diverse sulle prestazioni del modello.
Il primo requisito riguarda la necessità che le unità di misura delle metriche siano uniformi. Dato che RMSE, MAE, MAPE e MASE sono espresse tramite unità di misura differenti, è necessario che queste vengano prima normalizzate tramite una delle tecniche che discuteremo nella sezione successiva.
Un altro requisito chiave è l'assegnazione di pesi alle metriche. Dato che l'importanza delle metriche varia in base al contesto, è necessario stabilire un metodo per ponderare le metriche in maniera corretta.
Infine la metrica composita deve risultare interpretabile e robusta.

3.3 Normalizzazione delle metriche

Il primo problema che si presenta durante la definizione della metrica composita riguarda il fatto che le varie metriche in questione (RMSE, MASE, MAE, MAPE) vengono rappresentate tramite unità di misura differenti tra loro. RMSE e MAE sono metriche espresse utilizzando la stessa unità di misura dei dati originali, MAPE viene espressa tramite percentuale e infine MASE è una misura normalizzata. Queste differenze rendono impegnativo il loro confronto e complicata la loro combinazione. Diventa necessario quindi convertirle in modo da avere tutte le metriche rappresentate tramite un'unità comune. Per fare ciò, deve essere applicata la tecnica di validazione incrociata, che fornisce più valori di metriche per uno stesso modello, aumentando l'affidabilità della valutazione delle prestazioni dei modelli. Questo metodo suddivide il dataset in più parti, utilizzandone alcune per l'addestramento e altre per la valutazione garantendo una stima più robusta della bontà del modello. Esistono strategie di validazione incrociata pensate appositamente per i dati temporali, come verrà discusso nella sezione 4.3.

3.3.1 Normalizzazione Min-Max

Ai fini di questa tesi è stato adottato il metodo di normalizzazione Min-Max per scalare le metriche di errore in un intervallo compreso tra 0 e 1. Ogni valore della metrica viene trasformato associando il minimo a 0 e il massimo a 1, in modo tale che ogni altro valore presente tra questi due venga trasformato in un decimale tra

0 e 1. Ad esempio, se il valore minimo di una metrica fosse 20 e quello massimo 40, allora 30 verrebbe trasformato in 0.5 circa, dato che si trova a metà tra 20 e 40. Ai fini di questo progetto gli estremi da utilizzare per questa tecnica sono stati determinati basandosi sui valori delle metriche ottenuti durante il processo di cross-validation. L'utilizzo della normalizzazione min-max è motivato dalla necessità di avere dei valori compresi tra 0 e 1 in modo da poterli poi integrare nel calcolo della metrica composita senza ottenere valori negativi come risultato.

3.4 Scelta e composizione dei pesi delle metriche

Il peso da assegnare ad ogni metrica di errore è formato da 3 valori moltiplicati tra loro. Possiamo esprimerlo tramite la seguente formula:

$$w_i = F_s \cdot M_i^{-1} \cdot C_i$$

Dove:

- w_i è il peso della metrica di errore i-esima utilizzato all'interno della formula per il calcolo della metrica composita.
- F_s è la forza della stagionalità dei dati utilizzati, calcolata tramite decomposizione STL, in quanto modelli diversi possono avere performance che variano in base alla presenza più o meno marcata della stagionalità.
- M_i^{-1} è l'inverso della mediana o della media dei valori della metrica i-esima, scelto in base alla sensibilità agli outliers (media se rilevanti, mediana se anomalie), poiché permette di normalizzare l'impatto della metrica e ridurre l'influenza di valori estremi, garantendo una valutazione più robusta e rappresentativa della prestazione del modello.
- C_i è la componente di significatività che tiene conto delle caratteristiche specifiche della metrica i-esima e della loro rilevanza rispetto a valori di stagionalità, trend e outliers del dataset in uso.

3.4.1 Calcolo della forza di tendenza e stagionalità

La forza della tendenza (F_T) e la forza della stagionalità (F_S) sono valori fondamentali per la creazione della metrica composita. Il valore F_S è direttamente coinvolto nel calcolo del peso delle varie metriche, dato che rappresenta l'influenza della stagionalità nel dataset e, di conseguenza, influenza l'importanza relativa di ciascuna metrica di errore. Poiché modelli diversi possono avere prestazioni differenti in base alla stagionalità dei dati, includere F_S nel calcolo del peso permette di dare maggiore importanza alle metriche che meglio descrivono l'errore in contesti altamente stagionali e di ridurne l'impatto quando la stagionalità è meno pronunciata. Inoltre, il calcolo delle componenti F_T e F_S è utile anche per determinare i coefficienti di significatività C_i, i quali rappresentano il peso relativo di ciascuna metrica di errore nel contesto del dataset specifico. Le due componenti sono definite come segue.

Forza della tendenza (F_T)

La forza della tendenza misura quanto il trend è predominante rispetto alla parte della serie temporale che non è stagionale. Viene calcolata tramite la formula:

$$F_T = max\left(0, 1 - \frac{Var(R_t)}{Var(T_t + R_t)}\right)$$

Ciò fornirà una misura della forza del trend tra 0 e 1. Poiché la varianza del resto potrebbe occasionalmente essere anche maggiore della varianza dei dati destagionalizzati, impostiamo il valore minimo possibile di F_T uguale a zero. Un valore vicino a 1 indica un trend dominante, valori bassi invece suggeriscono che il trend ha poca influenza rispetto alle altre componenti.

Forza della stagionalità (F_S)

La forza della stagionalità è definita in maniera simile a quella precedente ma rispetto ai dati detrended piuttosto che ai dati destagionalizzati:

$$F_S = max\left(0, 1 - \frac{Var(R_t)}{Var(S_t + R_t)}\right)$$

Una serie con forza stagionale F_S vicino a 0 sarà una serie che presenta poca stagionalità nei dati, mentre una serie con una forte stagionalità avrà F_S vicino ad 1 dato che $Var(R_t)$ sarà molto più piccola di $Var(S_t + R_t)$.[1]

3.4.2 Calcolo della media o mediana dei valori della metrica

Un'altra componente importante nel calcolo dei pesi da assegnare alle varie metriche dipende dal dataset che stiamo analizzando. Durante il calcolo dei pesi è importante capire se gli outliers presenti nel dataset sono importanti o meno per l'analisi. Se i valori anomali sono da considerarsi utili, come ad esempio nel caso di un analisi clinica, allora la componente verrà calcolata utilizzando la media dei valori della metrica. Nel caso contrario, quindi in una situazione in cui i valori anomali disturbano l'analisi, viene utilizzata la mediana come componente facente parte del peso della metrica.

Utilizzo della media

La media aritmetica è definita come la somma di un insieme di valori divisa per il numero totale di elementi. Nel contesto di questa tesi, per ogni modello viene calcolata la media delle performance di ogni metrica utilizzando i risultati ottenuti tramite le varie iterazioni di cross-validation. La media consente di ottenere un valore rappresentativo della performance complessiva del modello, tenendo conto di tutte le osservazioni.
A differenza della mediana, la media è più sensibile alla presenza di outliers, il che può essere sia un vantaggio che uno svantaggio a seconda del contesto. Se gli outliers rappresentano errori di misura o anomalie indesiderate, l'uso della media potrebbe distorcere la valutazione complessiva. Tuttavia, in alcuni casi gli outliers possono

[1] X. Wang, Smith e R. Hyndman 2006.

essere informazioni utili che riflettono particolari condizioni critiche di previsione del modello. In queste situazioni, la media consente di catturare l'effettiva variabilità della performance del modello, offrendo una visione più completa rispetto alla mediana.
La formula utilizzata per determinare il peso del tipo di modello rispetto a una determinata metrica è data da:

$$V_m = \frac{1}{Mean_m}$$

Dove:

- V_m è il valore inverso della media dei valori della metrica m.
- $Mean_m$ è la media dei valori della metrica m ottenuti nei diversi fold della *cross-validation*.

L'uso di $\frac{1}{Mean}$ nella determinazione dei pesi permette di dare maggiore enfasi alle metriche con errori medi più bassi, assegnando loro un peso maggiore nella valutazione complessiva. Questo approccio è utile in contesti in cui è necessario valutare l'affidabilità del modello considerando l'intera distribuzione degli errori, inclusi eventuali valori estremi. Utilizzare la media al posto della mediana risulta particolarmente vantaggioso quando gli *outliers* sono informativi e possono indicare situazioni critiche di previsione che non devono essere trascurate. In questo modo, il metodo consente di pesare le metriche in modo più aderente alla realtà del fenomeno analizzato, senza eliminare informazioni potenzialmente rilevanti.

Utilizzo della mediana

La mediana è definita come il valore che si trova al centro di un'insieme di valori posti in ordine crescente. Nel contesto di questa tesi, per ogni modello viene calcolata la mediana delle performance di ogni metrica, utilizzando i risultati ottenuti tramite le varie iterazioni di *cross-validation*. Dal momento che la mediana rappresenta il valore centrale delle performance, essa è meno sensibile agli *outliers* rispetto alla media. Un modello che mostra una mediana bassa su una metrica (ad esempio, un RMSE basso) indicherà una performance costantemente buona su quel parametro. In tal caso, alla metrica verrà assegnato un peso maggiore, poichè è più rilevante nel determinare l'efficacia del modello. Al contrario, se la mediana della metrica è alta, il peso sarà ridotto, indicando che il modello non ottiene risultati ottimali su quella metrica specifica.
La formula utilizzata per determinare il peso del tipo di modello rispetto a una determinata metrica è data da:

$$V_m = \frac{1}{Med_m}$$

Dove:

- V_m è il valore inverso della mediana dei valori della metrica m.
- Med_m è la mediana dei valori della metrica m ottenuti nei diversi *fold* della *cross-validation*.

L'uso di $\frac{1}{Med}$ al posto della mediana stessa nel calcolo dei pesi ha una ragione importante legata al modo in cui si vogliono influenzare i pesi assegnati alle diverse metriche. Utilizzare la formula $\frac{1}{Med}$ permette di dare maggiore peso alle metriche con valori più bassi e ridurre allo stesso tempo il peso di metriche con valori più alti. In questo modo, riusciamo sia a rendere più influenti metriche con errori più piccoli (come ad esempio valori bassi di MAE o RMSE) che diminuire l'importanza di metriche con valori più elevati (e potenzialmente meno rappresentative della bontà del modello). Questo metodo consente di bilanciare meglio l'influenza delle diverse metriche, evitando che quelle con scale di valori naturalmente più alte dominino il calcolo della metrica composita.

3.4.3 Calcolo della componente di significatività delle metriche

La componente di significatività C_i è una parte fondamentale del peso da assegnare ad ogni metrica. Essa tiene conto delle caratteristiche di ogni metrica e ne combina i vari pregi e difetti in modo da creare un valore significativo in base alle caratteristiche del dataset in uso. Il criterio di assegnazione dei pesi è stato definito analizzando :

- Stagionalità nei dati : se la serie temporale presenta una elevata stagionalità, il MASE, che tiene conto del comportamento stagionale dei dati, sarà più importanti rispetto a metriche come RMSE che sono meno sensibili alla stagionalità e quindi avranno un peso maggiore.
- Trend nei dati : come per la stagionalità, anche in presenza di trend ci sono metriche che possono risultare più affidabili, ad esempio il MAE, che in presenza di trend dominante avrà un peso maggiore rispetto alle altre metriche
- Valori anomali : se nel dataset sono presenti *outliers* importanti o si vuole dare una maggiore importanza ad errori grandi, l'RMSE riceverà un peso maggiore rispetto alle altre metriche. Al contrario se si preferisce una valutazione più equa degli errori è da favorire il peso del MAE nel calcolo.
- Valori nulli : se all'interno del set di dati sono presenti valori nulli non sostituibili allora il MAPE viene automaticamente escluso assegnandogli un peso nullo, in modo da evitare problematiche nel calcolo dei risultati

Grazie all'analisi di questi punti è possibile adattare la metrica in base alle caratteristiche del dataset, aumentando la precisione della valutazione delle performance dei modelli.

3.5 Calcolo della metrica unica

La metrica unica composita, denominata CFEM (*Composite Forecasting Evaluation Metric*) viene calcolata effettuando una media pesata delle varie metriche di errore (RMSE, MAE, MAPE, MASE) moltiplicate per il loro relativo peso. La formula matematica per il calcolo è la seguente:

$$CFEM = \frac{\sum_{i=1}^{n} w_i * m_i}{\sum_{i=1}^{n} w_i}$$

Dove :

- n : è il numero delle metriche analizzate
- w_i : è il peso della i-esima metrica
- m_i : è il valore normalizzato della i-esima metrica

La normalizzazione delle metriche è stata effettuata tramite *min-max normalization*. Come descritto nella sezione 2.6.1, questa tecnica scala ogni valore in un intervallo [0,1], permettendo di bilanciare meglio il contributo di ogni metrica ottenendo un risultato più facile da interpretare e confrontare tra modelli.

Interpretazione della metrica

La CFEM rappresenta le performance dei modelli tramite un valore che indica l'errore predittivo combinato delle varie metriche, quindi più questo valore è basso migliori saranno le prestazioni del modello. L'introduzione dei pesi all'interno della metrica permette di catturare in maniera più accurata le prestazioni, attenuando eventuali criticità dimostrate dalle metriche singole. Ad esempio in presenza di serie temporali con una stagionalità elevata è importante attribuire un peso maggiore alle metriche in grado di catturare tali variazioni. Lo stesso accade con i valori anomali, in quanto la metrica permette di bilanciare l'impatto che questi hanno sulla valutazione complessiva.

Validazione della metrica

La fase di validazione è essenziale per confermare l'affidabilità della metrica composita in quanto a valutazione delle performance dei modelli di previsione. Per validare la CFEM è stato necessario confrontare i valori ottenuti dalla metrica composita con quelli derivanti dall'analisi delle metriche singole, in questo modo è stato possibile constatare che il ranking dei modelli basato sulle performance predittive delle metriche rimane lo stesso anche osservando i valori delle varie CFEM. Inoltre è stata svolta un'analisi riguardo al cambio dei pesi che, rimanendo entro un range ragionevole di valori (ad esempio modifiche inferiori al 20% rispetto ai valori iniziali), non altera troppo il risultato dei modelli rendendo la metrica stabile anche in caso di assegnamento dei pesi non troppo preciso.

4 Implementazione e sperimentazione

4.1 Panoramica dei dataset

In questa sezione sono racchiuse tutte le informazioni relative ai due set di dati utilizzati durante lo svolgimento di questo progetto. Il primo dataset, *Hourly Energy Consumption* è stato scelto tenendo conto della necessità di avere una buona stagionalità dei dati nel tempo, effettuando una ricerca sulla piattaforma Kaggle, ossia un portale che fornisce la possibilità di scaricare gratuitamente vari tipi di dataset. Il secondo set di dati proviene direttamente dall'azienda che ha commissionato questo progetto ed è composto da dati di clienti reali.

4.1.1 Dataset "Hourly Energy Consumption"

Questo set di dati proviene da un'azienda statunitense, la PJM Interconnection LLC, che si occupa della trasmissione elettrica regionale (RTO) di alcune regioni degli Stati Uniti. Il set è composto da dati relativi a quindici anni di consumo elettrico orario delle varie regioni, espressi in megawatt (MW).

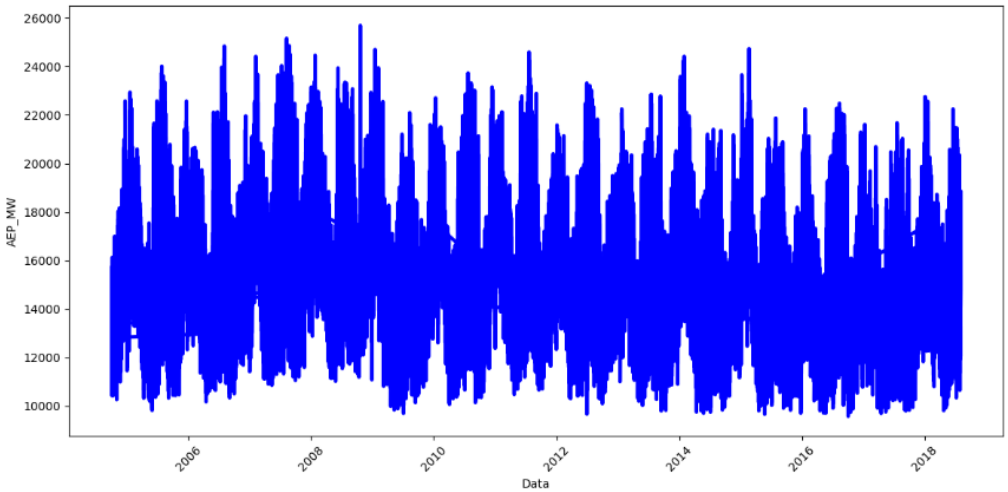

Figura 4.1: Visualizzazione del dataset "Hourly Energy Consumption" completo

Grazie al grafico stampato utilizzando la libreria MatPlotLib di Python è possibile analizzare l'andamento dei dati di questo dataset. Si nota una chiara stagionalità annuale, caratterizzata da picchi regolari nel consumo di energia in corrispondenza dei mesi più caldi e freddi dell'anno. La serie appare relativamente stabile nel tempo, con un livello generale di consumo energetico che non mostra variazioni di trend

significative. Sebbene nel grafico della serie temporale si osservino alcuni picchi, questi non rappresentano veri e propri outliers, ma riflettono invece un comportamento atteso del fenomeno analizzato. In particolare, trattandosi di dati sul consumo elettrico, i valori elevati corrispondono ai giorni con temperature più estreme, in cui il fabbisogno energetico aumenta per l'uso di climatizzatori o riscaldamento. Pertanto, non si è ritenuto necessario applicare un trattamento specifico per questi valori, in quanto fanno parte della naturale variabilità della serie e contribuiscono alla sua stagionalità.

4.1.2 Dataset aziendale "Humco"

In questo dataset sono presenti dati relativi alla vendita di un prodotto da parte di un'attività cliente dell'azienda *Humco*. L'azienda si è occupata di rimuovere le informazioni sensibili dei compratori, permettendo di utilizzare il set di dati per lo sviluppo del progetto. Il risultato ottenuto è un dataset composto da una colonna che riporta le vendite giornaliere di un dato prodotto in una finestra temporale di sei anni. Il dataset non contiene valori mancanti ma sono presenti molteplici valori nulli, indicativi di una giornata in cui non è stato venduto nessun prodotto.

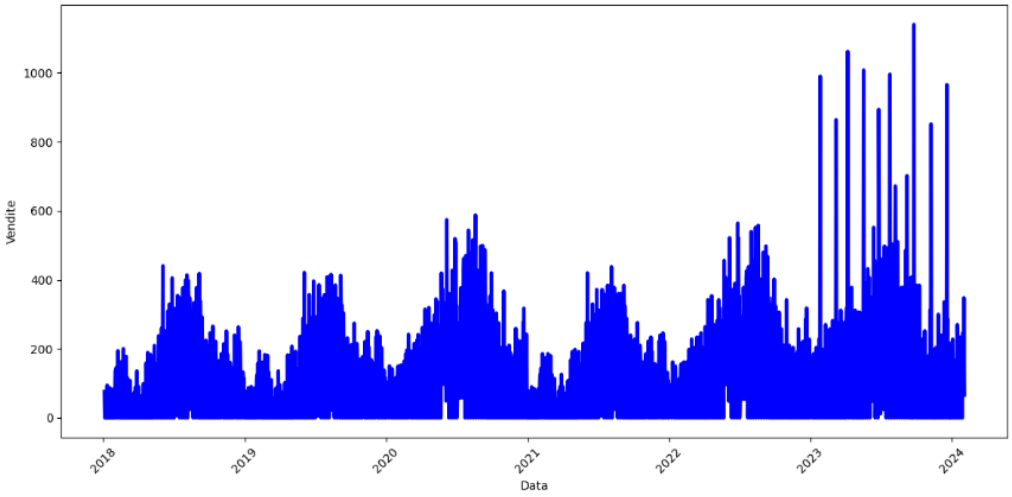

Figura 4.2: Visualizzazione del dataset aziendale completo

Tramite il grafico riportato qui sopra possiamo notare che la stagionalità dei dati è meno marcata del dataset precedente, mostrando andamenti ciclici che suggeriscono variazioni periodiche nell'attività aziendale. A differenza del primo dataset, si nota una maggiore variabilità nella parte più recente della serie, con un aumento significativo dei valori massimi dovuto alla presenza di picchi improvvisi e molto pronunciati. Questo potrebbe essere attribuito a cambiamenti all'interno dell'azienda o ad eventi anomali.

4.2 Metodi di preprocessing

Questa sezione contiene la spiegazione delle modifiche apportate ai dati prima dell'analisi tramite modello predittivo.

Dataset "Hourly Energy Consumption"

Per garantire una corretta analisi sono state apportate le seguenti modifiche ai dati:

1. Conversione della colonna temporale : La colonna relativa alla data espressa in timestamp è stata convertita in un oggetto di tipo DateTimeIndex in modo da facilitare l'utilizzo dei modello di previsione che lavorano con serie temporali

2. Ordinamento dei dati : I dati sono stati ordinati cronologicamente per garantire una corretta analisi temporale

3. Raggruppamento giornaliero : Per ridurre il numero di osservazioni, i dati sono stati aggregati a livello giornaliero, calcolando la media del consumo elettrico per ciascun giorno.

4. Filtraggio dell'intervallo temporale : Per questo progetto è stato considerato un sottoinsieme del dataset originale, composto da dati che vanno dall'anno 2008 all'anno 2017 compreso.

5. Impostazione della colonna Datetime come indice : Alcuni modelli di previsione come in questo caso ARIMA e Prophet, richiedono che la colonna relativa alla data sia l'indice del dataframe in modo da migliorare la gestione dei dati

6. Assegnazione di una variabile target : Per il corretto utilizzo del modello Prophet è necessario che la variabile target sia contenuta all'interno di una variabile `y`

Dataset aziendale *Humco*

Per quanto riguarda la fase di preprocessing di questo dataset possiamo dire che sono state effettuate anche qui le modifiche riportate nella sezione precedente relative al primo set di dati con l'aggiunta della gestione di alcuni outliers visualizzati tramite *scatter plot* un tipo di grafico che permette una visualizzazione dei dati tramite punti.

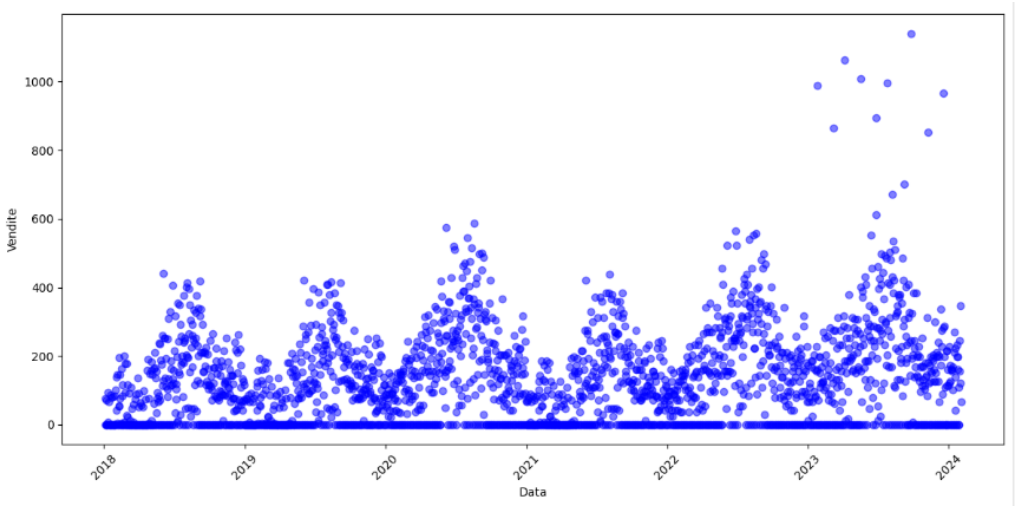

Figura 4.3: Visualizzazione tramite "scatter plot" del dataset aziendale completo

Gli outliers che si intravedono sono concentrati durante l'anno 2023. É stato possibile individuarli tramite il metodo dell'IQR (*Interquartile Range*) che permette di definire una soglia oltre la quale un valore viene considerato anomalo. Per prima cosa vengono calcolati il primo quartile (Q1, il 25° percentile) e il terzo quartile (Q3, il 75° percentile) e successivamente viene definito il range IQR come differenza tra i due quartili appena calcolati ($IQR = Q3 - Q1$). Infine si stabilisce un intervallo accettabile, definito come :

- Limite inferiore : Q1 - 1.5 * IQR
- Limite superiore : Q3 + 1.5 * IQR

Così facendo vengono individuati i valori che si trovano al di fuori di questo intervallo e successivamente vengono sostituiti con la mediana della serie temporale in modo da mantenere la robustezza.

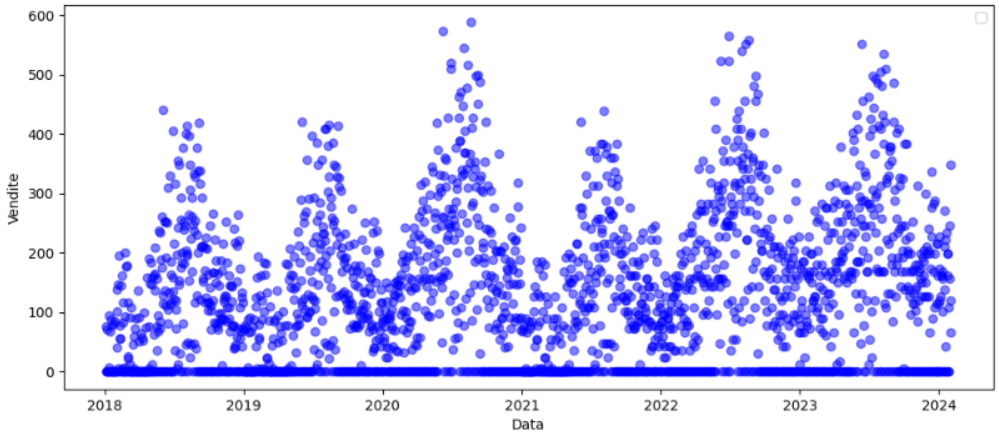

Figura 4.4: Visualizzazione tramite *scatter plot* del dataset dopo la rimozione degli outliers

4.3 Expanding Window Cross-Validation

Prima di parlare dell'implementazione dei modelli è necessario definire quella che è stata la tecnica utilizzata per valutare le performance di ogni modello. Quando si lavora con serie temporali, i metodi tradizionali di convalida incrociata potrebbero non essere adatti a causa della natura temporale dei dati. Ad esempio, la convalida incrociata *k-fold* non preserva l'ordine temporale dei dati e potrebbe stimare le prestazioni in maniera errata. Nel caso di questo progetto è stata utilizzata la tecnica di convalida incrociata a finestra espandibile (*Expanding Window Cross-Validation*). La peculiarità di questo metodo deriva dal fatto che la finestra di training, cioè la parte di dati dedicata all'addestramento del modello, non ha una misura fissa, inizia con una dimensione piccola e si espande gradualmente per includere più punti dati. Il modello viene addestrato su ogni finestra e valutato su quella successiva.
Un'alternativa comune è la *Sliding Window Cross-Validation*, in cui la finestra di training ha una dimensione fissa e si sposta lungo la serie temporale. Tuttavia, questa tecnica può risultare meno efficace quando i dati più vecchi contengono ancora informazioni utili alla previsione, poiché la finestra fissa esclude progressivamente le osservazioni più datate. Ai fini di questo progetto, mantenere l'intera cronologia dei dati di training ha permesso di sfruttare meglio la struttura temporale della serie,

migliorando la capacità del modello di adattarsi a trend e stagionalità di lungo periodo.
Possiamo concludere affermando che l'applicazione di questa tecnica è fondamentale per valutare le prestazioni dei modelli predittivi e, considerando l'ordinamento temporale dei dati, fornisce stime di prestazioni più realistiche[1].

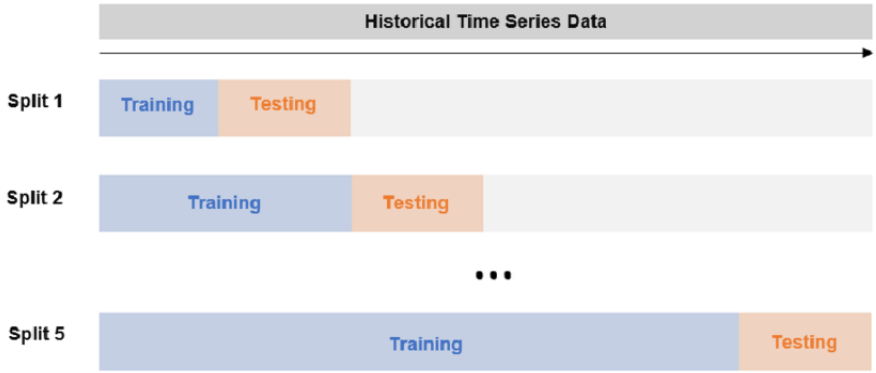

Figura 4.5: Esempio di convalida incrociata a finestra espandibile con 5 split (Fonte: Vien et al. 2021)

Implementazione nel caso studio

L'impostazione della tecnica è avvenuta basandosi sul set di dati analizzato. Dato che il primo dataset presenta una finestra temporale di dieci anni, e la stagionalità dei dati risulta essere all'incirca di 6/12 mesi, la dimensione della finestra di train è stata settata a 365 giorni, corrispondente ad un anno di dati. Questo significa che ogni modello viene valutato su un periodo annuale. Essendo la tecnica a finestra espandibile, il primo train set cresce di 365 giorni per ogni fold, finchè non arriva ad includere quasi tutto il dataset a fine validazione. Questo permette di avere la garanzia che l'apprendimento del modello cresca in proporzione alla mole di dati, assumendo un comportamento realistico di aggiornamento del modello nel tempo.
Il numero di fold viene calcolato come $n\_splits = len(y) \,//\, n\_test\_years$ cioè il numero massimo di suddivisioni possibili considerando la lunghezza della serie. L'ultimo split è stato volutamente escluso dalla validazione incrociata poichè viene riservato al test finale.
In conclusione all'analisi viene generato un grafico con i fitted values e la previsione ottenuta dal modello, confrontando questi dati con quelli reali. Grazie a ciò possiamo avere un riscontro visivo di quella che è stata la previsione fornita dal modello.

4.4 Modelli utilizzati

Durante lo sviluppo di questo progetto sono stati utilizzati quattro modelli: ARIMA, SARIMA, Holt-Winters e Prophet. Sono stati implementati utilizzando le relative librerie Python:

- *statsmodels.tsa.arima.model* per il modello ARIMA

[1]Palvel 2023.

- *statsmodels.tsa.statespace.sarimax* per il modello SARIMA
- *statsmodels.tsa.holtwinters* per il modello ARIMA
- *prophet* per il modello Prophet

Come citato nella sezione 2.4, sono modelli che possiedono caratteristiche diverse che si comportano in maniera differente a seconda del tipo di dataset che si sta analizzando.

4.4.1 Modello ARIMA

Per utilizzare correttamente il modello ARIMA è necessario settare con criterio i parametri *(p, d, q)* del modello. Esistono tecniche di selezione, come ad esempio ACF, PACF e *auto_arima*, che permettono di trovare i parametri migliori, ma queste non funzionano sempre al meglio.
Nel caso del primo dataset i parametri sono stati scelti testando diverse combinazioni di valori e scegliendo quella che minimizzava le metriche di errore, dato che, in questo caso, la selezione manuale ha prodotto dei risultati migliori in termini di accuratezza predittiva rispetto all'utilizzo di tecniche come *auto_arima*, testata inizialmente. I risultati migliori in termini di metriche di errore sono stati utilizzando *(p=5, d=0, q=4)* come parametri per il modello.
Riguardo il dataset aziendale invece, l'utilizzo della tecnica di ricerca automatica dei parametri migliori, *auto_arima*, si è rivelato conveniente in quanto manualmente non è stato possibile trovare parametri che riuscissero a minimizzare le metriche di errore maggiormente rispetto a quelli trovati automaticamente. Questi risultano essere (p=5, d=1, q=5).
In entrambi i dataset il rischio di possibile *overfitting* è stato ridotto grazie all'applicazione della tecnica di validazione incrociata, effettuata utilizzando una finestra di 365 giorni come dimensione.

4.4.2 Modello SARIMA

Come riportato nella sezione 2.4.2 il modello SARIMA è un'estensione del modello ARIMA, con la differenza che è progettato per gestire dati che presentano una stagionalità. Questo modello quindi possiede gli stessi parametri di ARIMA spiegato nella sezione precedente, con l'aggiunta di valori utili ad identificare la componente stagionale dei dati.
Per ragioni di limitata potenza di calcolo non è stato possibile utilizzare l'algoritmo *auto_arima* per trovare la combinazione migliore dei parametri. É stato necessario quindi ricorrere all'utilizzo dell'algoritmo FFT(*Fast Fourier Transform*) descritto alla *sezione 2.6* per trovare, e successivamente testare, dei possibili valori di stagionalità da assegnare al parametro s del modello SARIMA. I parametri rimanenti (*P, D, Q*) sono stati testati manualmente e scelti secondo la combinazione migliore in termini di bontà predittiva.
I parametri del modello SARIMA per il primo dataset risultano essere :

- Parametri non stagionali : p=5, d=0, q=4
- Parametri stagionali : P=1, D=1, Q=1, s=30 (lunghezza del ciclo stagionale in giorni)

Mentre per il secondo dataset, quello aziendale:

- Parametri non stagionali : p=5, d=1, q=5
- Parametri stagionali : P=1, D=1, Q=1, s=36 (lunghezza del ciclo stagionale in giorni)

4.4.3 Modello Holt-Winters

Come riportato nella sezione 2.4.3, esistono due varianti di questo modello: la variante additiva, preferibile quando le variazioni stagionali sono tendenzialmente costanti nella serie e la variante moltiplicativa, preferibile quando le variazioni stagionali cambiano con il livello della serie temporale[2]. É possibile scegliere il tipo di variante da utilizzare tramite i parametri *trend* e *seasonal*. Dopo aver eseguito entrambe le varianti, per il primo set di dati risulta essere migliore in termini di accuratezza predittiva il modello moltiplicativo, con periodo di stagionalità uguale a 180 giorni. Per il secondo dataset, vista la presenza di valori nulli, l'unica variante utilizzabile è quella additiva. Il valore migliore per il parametro relativo alla stagionalità risulta essere 60, che restituisce risultati migliori in termini di accuratezza. Anche in questo caso sono stati testati i periodi migliori trovati tramite FFT.

4.4.4 Modello Prophet

Per eseguire correttamente il modello Prophet è necessario rinominare le colonne del dataset che stiamo analizzando. La colonna relativa alla data dovrà essere rinominata in *ds* mentre la colonna della variabile target dovrà essere indicata con la lettera *y*. Analogamente ai modelli precedenti, anche per Prophet sono stati testati diversi parametri per migliorare l'accuratezza delle previsioni.
Dopo alcune prove, la combinazione dei parametri *yearly_seasonality=True* e *daily_seasonality=False* risulta essere quella che restituisce risultati migliori in termini di accuratezza su entrambi i dataset. Altri parametri come ad esempio *changepoint_prior_scale* non apportano miglioramenti alla predizione e per questo motivo sono stati lasciati al valore di default. Come per gli altri modelli è stata utilizzata la tecnica di validazione incrociata con finestre di 365 giorni per ridurre il rischio di *overfitting*.

[2]R. J. Hyndman e Athanasopoulos 2018.

5 Risultati e Valutazione

Questo capitolo racchiude tutti i risultati ottenuti durante lo sviluppo di questo progetto. Include un confronto dei valori delle metriche ottenute tramite l'esecuzione dei vari modelli, per entrambi i dataset.

5.1 Confronto dei modelli

5.1.1 Risultati ottenuti analizzando il dataset "Hourly Energy Consumption"

Per quanto riguarda l'analisi del dataset relativo ai consumi elettrici statunitensi, abbiamo riscontrato dei risultati abbastanza soddisfacenti utilizzando i modelli di previsione.
I valori ottenuti effettuando la media delle metriche ottenute ad ogni fold sono stati trascritti nella seguente tabella:

Metrica	**Holt-Winters**	**Prophet**	**ARIMA**	**SARIMA**
RMSE	1884.27	1404.05	2002.35	1935.15
MAE	1540.19	1087.85	1672.24	1606.04
MAPE	10.01	6.97	11.34	10.80
MASE	0.0977	0.0687	0.1056	0.1015

Tabella 5.1: Valori medi delle metriche di previsione per i vari modelli su dataset statunitense.

É possibile notare come il modello Prophet è quello che ottiene risultati migliori in termini di metriche, a seguire il modello Holt-Winters ed infine i modelli SARIMA e ARIMA che ottengono risultati simili. Questi risultati possono essere giustificati dal fatto che Prophet gestisce in modo automatico la stagionalità e il trend, adattandosi meglio alle caratteristiche della serie. Inoltre Holt-Winters è efficace per serie con stagionalità regolare come questa ed ARIMA e SARIMA potrebbero essere penalizzati dalla necessità di un'accurata selezione dei parametri.
A seguito del termine dell'esecuzione di ogni modello viene applicata la normalizzazione min-max in modo da portare tutte le metriche su una scala unica.

Metrica	Holt-Winters	Prophet	ARIMA	SARIMA
RMSE	0.3714	0.2944	0.3752	0.3458
MAE	0.3224	0.2571	0.3535	0.3218
MAPE	0.3338	0.2923	0.3574	0.3161
MASE	0.3258	0.2568	0.4152	0.3382

Tabella 5.2: Valori medi delle metriche di errore su dataset statunitense, ottenuti tramite normalizzazione min-max.

I risultati della normalizzazione riportati nella tabella soprastante consentono un confronto più equo tra le metriche, dato che queste sono tutte sulla stessa scala numerica.
Analizzando i valori normalizzati, Prophet continua ad essere il modello con le prestazioni migliori, avendo i valori più bassi per tutte le metriche. Tuttavia, rispetto alla prima tabella, si nota una riduzione delle differenze tra i modelli. Ad esempio, nella tabella originale il RMSE di Holt-Winters e SARIMA differiva di circa 50 unità (1884.27 vs 1935.15), mentre nella tabella normalizzata la differenza è solo di 0.0264 (0.3714 vs 0.3458). Questo effetto è dovuto alla compressione dei valori all'interno di un intervallo ristretto, che riduce la percezione delle differenze assolute tra i modelli.
Un aspetto interessante è che il ranking dei modelli rimane invariato rispetto ai valori originali: Prophet mantiene il punteggio più basso in ogni metrica, seguito da Holt-Winters, mentre ARIMA e SARIMA ottengono risultati peggiori. Questo indica che la normalizzazione non ha alterato l'ordine di performance dei modelli, il che è un buon segnale della coerenza del processo di trasformazione. Tuttavia, la differenza tra i modelli meno performanti diventa meno evidente. Ad esempio, nella prima tabella ARIMA aveva il valore di MASE più alto (0.1056 contro 0.1015 di SARIMA), ma dopo la normalizzazione la differenza si amplifica (0.4152 vs 0.3382), suggerendo che la dispersione dei valori originali ha influenzato la normalizzazione in modo non uniforme. L'uso di pesi per le metriche nella metrica composita può mitigare questo problema distribuendo l'importanza in base alla rilevanza di ciascuna metrica nel contesto specifico del dataset.
In sintesi, la normalizzazione ha mantenuto il ranking dei modelli, ed il fatto che ha alterato la percezione delle differenze tra essi non è un problema dato che pesare le metriche in modo mirato permette di ridurre l'impatto delle distorsioni indotte dalla normalizzazione e di mantenere un confronto più fedele tra i modelli.

5.1.2 Risultati ottenuti analizzando il dataset aziendale

Per quanto riguarda il dataset fornito dall'azienda, i risultati in termini di metriche non sono buoni come quelli del dataset precedente.
Possiamo consultare i valori medi delle metriche nella tabella seguente:

Metrica	Holt-Winters	Prophet	ARIMA	SARIMA
RMSE	134.10	131.72	144.85	169.98
MAE	106.45	106.60	104.46	131.72
MAPE	inf	inf	inf	inf
MASE	0.9760	0.9787	0.9573	1.2353

Tabella 5.3: Valori medi delle metriche di errore per i vari modelli su dataset aziendale.

Notiamo subito che il calcolo della metrica MAPE restituisce come valore infinito, ciò è dato dal fatto che nel dataset sono presenti valori nulli, come già spiegato nella sezione 4.1.2. Questo valore è stato escluso dal calcolo della CFEM assegnando un peso uguale a zero alla metrica in questione. Basandoci sulle metriche rimaste possiamo affermare che i modelli Prophet e Holt-Winters performano leggermente meglio degli altri due, con il modello SARIMA che risulta essere il peggiore.
Come per il precedente dataset anche in questo caso analizziamo i risultati ottenuti dalla standardizzazione delle metriche tramite normalizzazione min-max.

Metrica	Holt-Winters	Prophet	ARIMA	SARIMA
RMSE	0.5027	0.5869	0.3710	0.4426
MAE	0.4998	0.5557	0.4598	0.4326
MAPE	nan	nan	nan	nan
MASE	0.4421	0.5040	0.4446	0.4344

Tabella 5.4: Valori medi delle metriche di errore su dataset aziendale, ottenuti tramite normalizzazione min-max.

Al contrario della situazione precedente, la normalizzazione non ha portato a risultati soddisfacenti, evidenziando alcune incongruenze rispetto al ranking dei modelli osservato nei valori originali.
Nella prima tabella (valori non normalizzati), Holt-Winters e Prophet risultavano i migliori modelli, con prestazioni simili tra loro, seguiti da ARIMA e infine da SARIMA, che aveva le peggiori performance. Questo si deduce dal fatto che RMSE e MAE sono più bassi per Prophet e Holt-Winters, mentre SARIMA presenta gli errori più alti. Tuttavia, nella seconda tabella con valori normalizzati, la situazione cambia: ARIMA, che prima risultava peggiore di Prophet e Holt-Winters, ora ha valori normalizzati più bassi per RMSE e MAE, suggerendo un miglior ranking rispetto a prima. Il modello Holt-Winters, che prima era tra i migliori, a seguito della normalizzazione ottiene valori più alti di ARIMA in tutte le metriche, mentre prima il suo RMSE e MAE erano inferiori a quelli di ARIMA. SARIMA, che nel confronto precedente era nettamente il peggiore, ha ora valori simili o migliori rispetto agli altri modelli nelle metriche normalizzate. Queste variazioni indicano che la normalizzazione ha alterato il ranking relativo dei modelli.

5.1.3 Confronto dei risultati della normalizzazione

Nel primo dataset, "Hourly Energy Consumption", la normalizzazione min-max non aveva causato queste incongruenze: i ranking tra i modelli erano rimasti invariati dopo la trasformazione. Questo suggerisce che il problema di incoerenza nella normalizzazione emerge solo nel dataset aziendale.
Possono essere molteplici le cause di questa incongruenza:

- Variabilità delle Metriche nel Dataset "Humco": Nel primo dataset le metriche hanno un range di variazione più proporzionato tra loro, quindi la normalizzazione ha mantenuto i rapporti originali tra i modelli. Nel dataset aziendale, invece, alcune metriche hanno variazioni più estreme, il che distorce il modo in cui i valori vengono scalati tra 0 e 1.
- Distribuzione dei valori delle metriche: Se alcune metriche presentano una distribuzione molto asimmetrica (per esempio, valori molto vicini tra loro per alcuni modelli e molto più alti per altri), la normalizzazione può comprimere eccessivamente alcune differenze e amplificare altre. Questo potrebbe spiegare perché ARIMA, che prima era inferiore a Prophet e Holt-Winters, ora appare più performante dopo la normalizzazione.
- Effetto della normalizzazione: Min-max normalizza ogni metrica indipendentemente, senza preservare le relazioni tra metriche diverse. Se una metrica varia di meno rispetto alle altre, la normalizzazione può ridurre la sua capacità discriminativa. Questo potrebbe spiegare perché SARIMA non sembra più così svantaggiato rispetto agli altri modelli nella tabella normalizzata.

Test di tecniche di normalizzazione differenti

Per provare ad ovviare a questo problema sono state testate altre tecniche di normalizzazione, tra cui: Robust min-max, trasformazione Sigmoidale, z-score, trasformazione logaritmica, e Robust scaling. Nessuna tra queste tecniche ha portato dei miglioramenti significativi in termini di coerenza tra le metriche normalizzate e quelle standard. Per questioni di praticità verranno riportati solo i risultati ottenuti tramite normalizzazione Robust min-max e Sigmoidale, tecniche definite alla sezione sezione 2.5

Normalizzazione min-max Robusta

Metrica	Holt-Winters	Prophet	ARIMA	SARIMA
RMSE	0.5140	0.5774	0.3998	0.4457
MAE	0.5174	0.5489	0.4831	0.4308
MAPE	nan	nan	nan	nan
MASE	0.4460	0.4950	0.4529	0.4328

Tabella 5.5: Valori medi delle metriche di errore su dataset aziendale, ottenuti tramite normalizzazione "Robust min-max".

Normalizzazione Sigmoidale

Metrica	Holt-Winters	Prophet	ARIMA	SARIMA
RMSE	0.5025	0.5062	0.4912	0.4949
MAE	0.5036	0.5044	0.4975	0.4936
MAPE	nan	nan	nan	nan
MASE	0.4930	0.5002	0.4926	0.4937

Tabella 5.6: Valori medi delle metriche di errore su dataset aziendale, ottenuti tramite normalizzazione "Robust min-max".

5.2 Analisi della scelta della componente di significatività C_i

La componente di significatività $C_i$descritta nella sezione 3.4.3 è un elemento chiave per la definizione dei pesi delle metriche di errore all'interno della CFEM. Questa componente viene assegnata a ciascuna metrica in base alle caratteristiche specifiche del dataset analizzato, come la forza della stagionalità, la presenza di trend e la distribuzione degli outlier. In questa sezione, analizzeremo il processo di assegnazione di C_i per i due dataset utilizzati in questo progetto. Verranno illustrate le motivazioni alla base delle scelte effettuate, mostrando come C_i sia stato modulato in funzione delle caratteristiche della serie temporale e del comportamento delle diverse metriche di errore.

Dataset Hourly Energy Consumption

La forza della stagionalità nel dataset è di 0.5759 e la forza del trend è di 0.8961, indicando un trend relativamente forte rispetto alla stagionalità. Inoltre, il dataset ha pochi outlier e una serie temporale regolare. Questi fattori influenzano la scelta dei pesi per ogni metrica di previsione.

Modello	RMSE	MAE	MAPE	MASE
C	0.25	0.30	0.20	0.25

Tabella 5.7: Valori della componente C_I selezionati in base alle caratteristiche del dataset statunitense

Possiamo giustificare la scelta dei valori come segue:

- RMSE (0.25) : Il RMSE è sensibile agli errori estremi, il che potrebbe non essere ideale per un dataset con pochi outliers come questo. Il peso assegnatogli è moderato in quanto mantiene una certa importanza per penalizzare gli errori più grandi in fase di previsione
- MAE (0.30) : Il MAE è meno sensibile agli errori estremi rispetto al RMSE e risulta quindi essere più adatto ad un dataset relativamente regolare come questo. Gli viene assegnato quindi un peso maggiore, grazie alla sua valutazione robusta della previsione

- MAPE (0.20) : Anche se il dataset non presenta valori bassi, il MAPE può comunque risultare meno utile in presenza di una serie temporale con trend marcato. Questo perché il MAPE misura l'errore in percentuale rispetto ai valori reali, e nei dataset con forte trend potrebbe non riflettere bene le performance previsionali rispetto ad altre metriche più dirette come MAE e RMSE. Per questo motivo, gli viene assegnato un peso inferiore rispetto alle altre metriche.

- MASE (0.25) : Considerando che la stagionalità presente nel dataset è buona, la metrica MASE offre un buon bilanciamento tra la previsione stagionale e quella complessiva. Gli viene assegnato un peso non troppo alto, ma comunque importante per tenere conto della stagionalità

Dataset aziendale "Humco"

La forza della stagionalità nel dataset è di 0.5365 e la forza del trend è di 0.5162, indicando una presenza moderata sia di stagionalità che di trend. Inoltre, gli outliers sono stati rimossi, rendendo il dataset più regolare. Questi fattori influenzano la scelta dei pesi per ogni metrica di previsione.

Modello	RMSE	MAE	MAPE	MASE
C	0.20	0.30	0.00	0.40

Tabella 5.8: Valori della componente C_i selezionati in base alle caratteristiche del dataset aziendale

Possiamo giustificare la scelta dei valori come segue:

- RMSE (0.20) : Poiché gli outliers sono stati rimossi, l'RMSE, che enfatizza gli errori più grandi, ha un peso inferiore rispetto ad altre metriche. Inoltre, la presenza moderata di trend e stagionalità non lo rende la metrica più adatta per catturare le dinamiche della serie.

- MAE (0.30) : Il MAE offre una misura più stabile dell'errore e, data la presenza di un trend moderato, viene valorizzato maggiormente rispetto all'RMSE. Tuttavia, il suo peso non è massimo perché il dataset presenta comunque stagionalità.

- MAPE (0.00) : Il dataset contiene valori uguali a zero, rendendo il MAPE inutilizzabile. Di conseguenza, gli viene assegnato un peso nullo.

- MASE (0.40) : La stagionalità nel dataset, pur non essendo fortissima, è comunque presente. Il MASE, essendo progettato per tenere conto della stagionalità nel calcolo dell'errore, diventa la metrica più rilevante, ricevendo quindi il peso maggiore.

Nonostante l'assegnazione dei pesi sia stata effettuata in base alle caratteristiche del dataset, resta evidente il problema legato alla normalizzazione delle metriche. Anche provando diverse variazioni di questi pesi il ranking dei modelli rimane alterato.

5.3 Analisi delle metriche standard e della metrica unica

In questa sezione vengono analizzati i risultati ottenuti dalle metriche standard di errore e dalla metrica unica composita CFEM per i diversi modelli di previsione utilizzati. L'obiettivo è valutare le prestazioni dei modelli sia tramite le metriche classiche che attraverso la CFEM per poter notare se i risultati complessivi delle prestazioni dei modelli coincidono in termini di ranking dei modelli.
Le metriche standard di errore (RMSE, MAE, MAPE e MASE) permettono di valutare le prestazioni dei modelli di previsione in modo individuale, mentre la CFEM fornisce una misura aggregata che combina queste metriche in un unico indice.

Dataset "Hourly Energy Consumption"

Nella seguente tabella sono ordinati i vari risultati ottenuti dall'analisi del dataset "Hourly Energy Consumption".

Modello	RMSE	MAE	MAPE	MASE	CFEM
Holt-Winters	1884.27	1540.19	10.01	0.0977	0.0435
Prophet	1404.05	1087.85	6.97	0.0687	0.0369
ARIMA	2002.35	1672.24	11.34	0.1056	0.0596
SARIMA	1935.15	1606.04	10.80	0.1015	0.0567

Tabella 5.9: Valori medi delle metriche di errore e della CFEM per modelli utilizzati su dataset statunitense.

Dall'analisi dei valori riportati in Tabella, si osserva che il ranking dei modelli ottenuto dalle metriche standard è coerente con quello restituito dalla CFEM:

- Prophet ottiene il miglior ranking sia considerando le metriche standard (minori valori di RMSE, MAE, MAPE e MASE) sia in base alla CFEM (0.0369, il valore più basso tra i modelli analizzati).
- Holt-Winters si posiziona al secondo posto in entrambi i confronti, avendo valori di errore inferiori rispetto ad ARIMA e SARIMA, ma superiori a Prophet. Anche la CFEM riflette questa posizione con un valore intermedio (0.0435).
- ARIMA e SARIMA risultano i modelli meno performanti in tutti i confronti. ARIMA ha i valori di errore più alti e una CFEM di 0.0596, mentre SARIMA presenta prestazioni leggermente migliori di ARIMA, ma comunque inferiori a Prophet e Holt-Winters, con una CFEM di 0.0567.

L'analisi mostra chiaramente che l'utilizzo della CFEM porta allo stesso ordinamento dei modelli ottenuto con le metriche standard. Questo conferma che la metrica unica rappresenta in modo coerente le informazioni fornite dalle metriche tradizionali, fornendo un'indicazione sintetica della qualità del modello rispettando l'ordine prestazionale dei modelli.

Dataset aziendale

Nella seguente tabella sono ordinati i vari risultati ottenuti dall'analisi del dataset aziendale.

Modello	RMSE	MAE	MAPE	MASE	CFEM
Holt-Winters	134.10	106.45	inf	0.9760	0.1179
Prophet	131.72	106.60	inf	0.9787	0.1343
ARIMA	144.85	104.46	inf	9573	0.1183
SARIMA	169.98	131.72	inf	1.2353	0.1157

Tabella 5.10: Valori medi delle metriche di errore e della CFEM per modelli utilizzati su dataset aziendale.

Dall'analisi dei valori riportati in Tabella, si osserva che il ranking dei modelli ottenuto dalle metriche standard non è coerente con quello restituito dalla CFEM. Anche variando la componente C del peso di ogni metrica nella composizione della CFEM l'ordinamento non corrisponde con quello relativo alle metriche tradizionali. Questa imprecisione è causata dalla normalizzazione delle metriche standard che non mantiene la classificazione dei modelli inalterata, come spiegato nella sezione 5.1.2.

5.4 Vantaggi e limitazioni della metrica composita

L'utilizzo della metrica composita CFEM risulta essere vantaggioso rispetto all'analisi delle varie metriche singole (RMSE, MAE, MAPE, MASE). Tra i vantaggi principali troviamo:

- Sintesi delle informazioni : La metrica composita riesce a racchiudere i risultati di quattro metriche in un unico valore, rendendo più facile e immediato il confronto delle prestazioni dei vari modelli. Questo è un grande vantaggio soprattutto quando si lavora con tanti modelli dato che si riducono le parti da analizzare.

- Riduzione della complessità : La complessità data dall'analisi di quattro metriche ognuna con la propria unità di misura viene ridotta grazie alla normalizzazione delle metriche. Il risultato è un unico valore in range da 0 a 1 che indica la bontà della predizione e grazie a questo si riesce a facilitare la comprensione dei risultati anche a persone non tecniche.

- Bilanciamento della valutazione : La differente valutazione data dalle varie metriche (es. RMSE per errori elevati, MAE per errori medi, MAPE per errori percentuali) viene bilanciata grazie all'applicazione della metrica composita CFEM, riducendo l'influenza di una singola prospettiva (ad esempio, gli outiler per RMSE)

- Flessibilità nei pesi : Grazie alla ponderazione è possibile assegnare priorità ad alcune metriche piuttosto di altre in base alla situazione in cui ci si trova,

adattandosi meglio alle esigenze specifiche del momento. Le singole metriche non offrono questa possibilità

L'utilizzo di questa metrica però comporta anche alcune limitazioni:

- Arbitrarietà pesi : La scelta dei pesi è decisamente impattante nel calcolo della metrica, nonostante vengano utilizzate informazioni derivate dai dati per effettuare la valutazione ci sarà sempre della soggettività nell'assegnazione dei pesi. Quindi questo è un fattore influenzato dalla competenza di chi effettua l'analisi
- Dipendenza dal dataset : La metrica potrebbe non funzionare altrettanto bene in contesti diversi, ad esempio utilizzando serie temporali con valori anomali non gestiti perchè magari importanti in quel contesto.
- Scalabilità : Nonostante la metrica funzioni bene in contesti come quelli visti in questo progetto, potrebbe essere meno efficace in contesti con serie altamente irregolari o con pattern non stazionari.

6 Conclusioni

6.1 Risultati finali

La metrica CFEM fornisce un valore indicativo dell'errore della previsione generata dal modello. Un valore basso indica che il modello ha una buona capacità di previsione in relazione agli errori combinati, mentre un valore alto suggerisce che il modello ha commesso errori significativi nel prevedere i dati. Inoltre la CFEM è pensata anche come metrica utile per comparare tra di loro vari modelli di previsione, basandosi sui loro errori complessivi. Quindi se il valore della metrica composita è più basso per un modello rispetto ad un altro significa che quel modello risulta essere più preciso, in termini di errore combinato, rispetto al secondo.

6.2 Limiti dello studio e sviluppi futuri

Lo sviluppo della metrica CFEM ha permesso di ottenere una misura sintetica dell'errore dei modelli di previsione, combinando diverse metriche tradizionali attraverso un approccio ponderato. I risultati ottenuti dimostrano che la CFEM può essere utilizzata per confrontare efficacemente modelli di forecasting, semplificando il processo di valutazione. Tuttavia, come qualsiasi metrica, anche la CFEM presenta alcune limitazioni, emerse durante l'analisi e la sperimentazione.
Questa sezione esamina i principali limiti dello studio e propone possibili sviluppi futuri per migliorare e ampliare l'applicabilità della metrica.

6.2.1 Limiti dello studio

Nonostante i buoni risultati ottenuti, il lavoro svolto presenta alcuni aspetti che potrebbero essere approfonditi. Di seguito esponiamo le limitazioni identificate durante lo studio.

Influenza della normalizzazione sulla classifica dei modelli

La CFEM richiede una fase di normalizzazione delle metriche per combinare valori con scale differenti. Tuttavia, è emerso che il metodo di normalizzazione utilizzato può influenzare il ranking finale dei modelli, portando a risultati non sempre coerenti con le metriche originali. Questo problema è stato riscontrato nel secondo dataset analizzato, dove alcune tecniche di normalizzazione hanno alterato la posizione relativa dei modelli rispetto a una valutazione basata sulle metriche standard.

Scelta dei pesi delle metriche

Un altro aspetto importante riguarda l'assegnazione dei pesi alle metriche nella CFEM. Sebbene siano stati introdotti criteri basati sulla forza della stagionalità e del trend per ridurre l'arbitrarietà della scelta, il processo di assegnazione dei pesi rimane in parte soggettivo. L'utilizzo di pesi fissi per ogni dataset implica che la CFEM potrebbe non adattarsi in modo ottimale a situazioni particolari, come serie temporali con comportamenti non convenzionali.

Validazione su un numero limitato di dataset

Lo studio ha testato la CFEM su due dataset con caratteristiche specifiche. Potrebbe essere utile una validazione più ampia su dataset di diversa natura (es. dati finanziari, biomedicali, meteorologici) per confermare la robustezza della metrica in contesti più vari.

Sensibilità a outlier e pattern particolari

I test effettuati hanno evidenziato che la CFEM potrebbe essere influenzata dalla presenza di outlier o pattern irregolari nella serie temporale. Anche se i valori anomali sono stati gestiti nel pre-processing, sarebbe utile verificare se la metrica mantiene la sua affidabilità in dataset con una maggiore presenza di anomalie o brusche variazioni nei dati.

6.2.2 Sviluppi futuri

Per affrontare le limitazioni evidenziate e migliorare l'applicabilità della CFEM, è necessario continuare ad integrare il progetto con nuovi sviluppi, come ad esempio:

Ottimizzazione della normalizzazione delle metriche

Un obiettivo futuro potrebbe essere l'analisi di metodi di normalizzazione più robusti, capaci di mantenere l'ordine relativo tra i modelli. Tecniche avanzate, come la normalizzazione basata su distribuzioni probabilistiche o su misure di robustezza (es. quantili, mediane), potrebbero ridurre l'impatto indesiderato della normalizzazione sui risultati finali.

Introduzione di pesi dinamici

Per migliorare la flessibilità della CFEM, una possibile evoluzione è l'implementazione di pesi dinamici che si adattino automaticamente alle caratteristiche del dataset. Un approccio basato su tecniche di machine learning potrebbe permettere di ottimizzare la combinazione delle metriche in base alla struttura della serie temporale, eliminando la necessità di una scelta manuale dei pesi.

Validazione su dataset più ampi e differenti

Un passo utile al miglioramento dell'affidabilità della CFEM, potrebbe essere il test della metrica su un maggior numero di dataset, possibilmente provenienti da ambiti diversi (energia, finanza, sanità, meteorologia). Un'analisi effettuata usando vari

tipi di set di dati potrebbe evidenziare eventuali debolezze della metrica e fornire indicazioni per migliorarla ulteriormente.

Studio della robustezza in presenza di outlier

Infine, un'ulteriore sviluppo potrebbe riguardare la valutazione della CFEM in presenza di serie temporali con forti outlier o trend non stazionari, per capire come la metrica si comporta in scenari più complessi e se sia necessario introdurre meccanismi di correzione per aumentarne l'affidabilità.

6.3 Conclusione finale

Grazie allo sviluppo di questo progetto è stato possibile definire e validare la CFEM (*Composite Forecasting Evaluation Metric*), una metrica progettata per fornire una valutazione sintetica e bilanciata dell'errore nei modelli di previsione delle serie temporali. Attraverso la combinazione ponderata di più metriche standard, la CFEM si propone come uno strumento utile per confrontare diversi modelli, tenendo conto delle caratteristiche di ognuno di essi della natura dei dati analizzati. I risultati ottenuti mostrano che la CFEM può offrire un'indicazione affidabile delle prestazioni dei modelli, contribuendo a semplificare il processo di selezione del miglior modello di previsione. Tuttavia, lo studio ha anche evidenziato alcune criticità, come l'influenza della normalizzazione e la soggettività nella scelta dei pesi, aspetti che potranno essere migliorati in futuri sviluppi della metrica. Le prospettive future suggerite in questa tesi aprono la strada ad ulteriori approfondimenti, con l'obiettivo di rendere la CFEM uno strumento ancora più robusto e applicabile in contesti reali.

A Codice Sorgente

A.0.1 Dataset "Hourly Energy Consumption"

Setup Dataset e preprocessing

```
1
2  # Import delle librerie utili
3  import pandas as pd
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from statsmodels.tsa.arima.model import ARIMA
7  from statsmodels.tsa.holtwinters import ExponentialSmoothing
8  from sklearn.metrics import mean_absolute_error, mean_squared_error
       , root_mean_squared_error
9  from statsmodels.tsa.statespace.sarimax import SARIMAX
10 from statsmodels.tools.sm_exceptions import ConvergenceWarning
11 from sklearn.metrics import mean_absolute_error
12 from prophet import Prophet
13 import warnings
14 from statsmodels.tsa.seasonal import STL
15
16
17 # Upload dataset su colab
18 from google.colab import files
19 uploaded = files.upload()
20 df=pd.read_csv('AEP_hourly.csv')
21
22
23 # Converti la colonna Datetime in DateTimeIndex e ordina la colonna
24 df['Datetime'] = pd.to_datetime(df['Datetime'], errors='coerce')  #
       Garantiamo la conversione
25 df_sorted = df.sort_values(by='Datetime').reset_index(drop=True)
26
27 # Raggruppa per giorno e calcola la media del consumo, quindi
       rinomina le colonne
28 df_daily = df_sorted.groupby(df_sorted['Datetime'].dt.date)['AEP_MW
       '].mean().reset_index()
29 df_daily.columns = ['Datetime', 'AEP_MW']
30
31 # Converti 'Datetime' in formato datetime se necessario
32 df_daily['Datetime'] = pd.to_datetime(df_daily['Datetime'], errors=
       'coerce')
33
34 # Filtra il DataFrame per gli anni da 2008 a 2017
35 df_subset_0817 = df_daily[(df_daily['Datetime'].dt.year >= 2008) &
       (df_daily['Datetime'].dt.year <= 2017)].copy()
36
```

```
# Crea una copia del dataset utile per il modello Prophet e imposta l'indice
df_prophet = df_subset_0817.copy()
df_subset_0817.set_index('Datetime', inplace=True)

# Verifica e converte l'indice in DatetimeIndex se necessario e imposta la frequenza giornaliera
if not isinstance(df_subset_0817.index, pd.DatetimeIndex):
df_subset_0817.index = pd.to_datetime(df_subset_0817.index)
df_subset_0817 = df_subset_0817.asfreq('D')

# Assegna alla variabile y il riferimento alla colonna AEP_MW
y = df_subset_0817['AEP_MW']

# Mostra i risultati
print(df_sorted.head())
print(df_sorted.tail())
print(df_daily)
print(df_subset_0817)
print(f"Frequenza dell'indice: {df_subset_0817.index.freq}")
```

```
pip install prophet --no-binary :all:
```

```
!pip install prophet
```

```
pip install pmdarima
```

Cross-Validation e modelli

```

# Ignora warning superflui
warnings.simplefilter('ignore', ConvergenceWarning)

def root_mean_squared_error(y_true, y_pred):
return np.sqrt(np.mean((y_true - y_pred) ** 2))

# Impostazioni per la cross-validation
n_test_years = 365
n_splits = len(y) // n_test_years

# Liste per raccogliere le metriche
metrics = {'HW': [], 'ARIMA': [], 'SARIMA': [], 'Prophet': []}

for i in range(n_splits - 1):
train_end = (i + 1) * n_test_years
train, test = y[:train_end], y[train_end:train_end + n_test_years]
train_prophet, test_prophet = df_prophet.iloc[:train_end], df_prophet.iloc[train_end:train_end + n_test_years]

models = {
        'HW': ExponentialSmoothing(train, trend='mul', seasonal='mul', seasonal_periods=182).fit(optimized=True),
        'ARIMA': ARIMA(train, order=(5, 0, 4)).fit(),
        'SARIMA': SARIMAX(train, order=(5, 0, 4), seasonal_order=(1, 1, 1, 30)).fit()
}
```

```
25
26 for model_name, model in models.items():
27 forecast = model.forecast(len(test))
28 rmse = root_mean_squared_error(test, forecast)
29 mae = mean_absolute_error(test, forecast)
30 mape = (abs(test - forecast) / test).mean() * 100
31 mase = mae / train.mean()
32 metrics[model_name].append((rmse, mae, mape, mase))
33 print(f"{model_name} - Fold {i+1}: RMSE={rmse:.2f}, MAE={mae:.2f},
    MAPE={mape:.2f}, MASE={mase:.4f}")
34 print("-" * 30)
35
36 # Prophet - Nuovo modello a ogni iterazione
37 prophet_model = Prophet(yearly_seasonality=True, daily_seasonality=
    False)
38 prophet_model.fit(train_prophet)
39
40 future = prophet_model.make_future_dataframe(periods=len(test),
    freq='D')
41 forecast_prophet = prophet_model.predict(future)['yhat'].iloc[-len(
    test):].values
42
43 rmse = root_mean_squared_error(test_prophet['y'].values,
    forecast_prophet)
44 mae = mean_absolute_error(test_prophet['y'].values,
    forecast_prophet)
45 mape = (abs(test_prophet['y'].values - forecast_prophet) /
    test_prophet['y'].values).mean() * 100
46 mase = mae / train_prophet['y'].mean()
47 metrics['Prophet'].append((rmse, mae, mape, mase))
48 print(f"Prophet - Fold {i+1}: RMSE={rmse:.2f}, MAE={mae:.2f}, MAPE
    ={mape:.2f}, MASE={mase:.4f}")
49 print("-" * 30)
50
51 # Grafico separato per ogni modello
52 for model_name, model in models.items():
53 plt.figure(figsize=(12, 6))
54 plt.plot(y.index, y, label='Original Data', color='blue')
55 plt.plot(test.index, model.forecast(len(test)), label=f'{model_name
    } Forecast', color='red')
56 plt.xlabel('Date'); plt.ylabel('AEP_MW')
57 plt.title(f'Forecast Last Fold - {model_name}')
58 plt.legend(); plt.xticks(rotation=45); plt.tight_layout()
59 plt.show()
60
61 # Prophet forecast plot separato (nuova istanza)
62 prophet_model = Prophet(yearly_seasonality=True, daily_seasonality=
    False)
63 prophet_model.fit(df_prophet.iloc[:train_end])
64 future = prophet_model.make_future_dataframe(periods=len(test),
    freq='D')
65 pred_prophet = prophet_model.predict(future)['yhat'].iloc[-len(test
    ):].values
66
67 plt.figure(figsize=(12, 6))
68 plt.plot(y.index, y, label='Original Data', color='blue')
69 plt.plot(test.index, pred_prophet, label='Prophet Forecast', color=
    'red')
```

```
plt.xlabel('Date'); plt.ylabel('AEP_MW')
plt.title('Forecast Last Fold - Prophet')
plt.legend(); plt.xticks(rotation=45); plt.tight_layout()
plt.show()

# Stampa medie e mediane delle metriche
for model_name, values in metrics.items():
values = np.array(values)
print(" ")
print(f"{model_name} - Media: RMSE={values[:,0].mean():.2f}, MAE={
    values[:,1].mean():.2f}, MAPE={values[:,2].mean():.2f}, MASE={
    values[:,3].mean():.4f}")
print(f"{model_name} - Mediana: RMSE={np.median(values[:,0]):.2f}, 
    MAE={np.median(values[:,1]):.2f}, MAPE={np.median(values[:,2])
    :.2f}, MASE={np.median(values[:,3]):.4f}\n")
print(" ")

# Estrarre le mediane delle metriche per ciascun modello
med_rmse_hw = np.median(np.array(metrics['HW'])[:, 0])
med_mae_hw = np.median(np.array(metrics['HW'])[:, 1])
med_mape_hw = np.median(np.array(metrics['HW'])[:, 2])
med_mase_hw = np.median(np.array(metrics['HW'])[:, 3])

med_rmse_arima = np.median(np.array(metrics['ARIMA'])[:, 0])
med_mae_arima = np.median(np.array(metrics['ARIMA'])[:, 1])
med_mape_arima = np.median(np.array(metrics['ARIMA'])[:, 2])
med_mase_arima = np.median(np.array(metrics['ARIMA'])[:, 3])

med_rmse_sarima = np.median(np.array(metrics['SARIMA'])[:, 0])
med_mae_sarima = np.median(np.array(metrics['SARIMA'])[:, 1])
med_mape_sarima = np.median(np.array(metrics['SARIMA'])[:, 2])
med_mase_sarima = np.median(np.array(metrics['SARIMA'])[:, 3])

med_rmse_prophet = np.median(np.array(metrics['Prophet'])[:, 0])
med_mae_prophet = np.median(np.array(metrics['Prophet'])[:, 1])
med_mape_prophet = np.median(np.array(metrics['Prophet'])[:, 2])
med_mase_prophet = np.median(np.array(metrics['Prophet'])[:, 3])
```

Normalizzazione min-max delle metriche ottenute

```

# Funzione di normalizzazione Min-Max
def min_max_normalization(value, min_value, max_value):
return (value - min_value) / (max_value - min_value) if max_value
    != min_value else 0

# Dizionario per raccogliere le metriche normalizzate
normalized_metrics = {}

for model_name in metrics.keys():
values = np.array(metrics[model_name])

min_rmse, max_rmse = values[:, 0].min(), values[:, 0].max()
min_mae, max_mae = values[:, 1].min(), values[:, 1].max()
min_mape, max_mape = values[:, 2].min(), values[:, 2].max()
min_mase, max_mase = values[:, 3].min(), values[:, 3].max()

```

```
normalized_rmse = [min_max_normalization(rmse, min_rmse, max_rmse)
    for rmse in values[:, 0]]
normalized_mae = [min_max_normalization(mae, min_mae, max_mae) for
    mae in values[:, 1]]
normalized_mape = [min_max_normalization(mape, min_mape, max_mape)
    for mape in values[:, 2]]
normalized_mase = [min_max_normalization(mase, min_mase, max_mase)
    for mase in values[:, 3]]

avg_normalized_rmse = np.mean(normalized_rmse)
avg_normalized_mae = np.mean(normalized_mae)
avg_normalized_mape = np.mean(normalized_mape)
avg_normalized_mase = np.mean(normalized_mase)

# Salva nel dizionario
normalized_metrics[model_name] = {
        'RMSE': avg_normalized_rmse,
        'MAE': avg_normalized_mae,
        'MAPE': avg_normalized_mape,
        'MASE': avg_normalized_mase
}

print(f"Media delle metriche normalizzate di tutti i fold ({
    model_name}):")
print(f"RMSE (normalizzato): {avg_normalized_rmse}")
print(f"MAE (normalizzato): {avg_normalized_mae}")
print(f"MAPE (normalizzato): {avg_normalized_mape}")
print(f"MASE (normalizzato): {avg_normalized_mase}")
print("-" * 60)
print(" ")

# Estrai i valori dal dizionario e assegna alle variabili
    necessarie
avg_normalized_rmse_hw = normalized_metrics['HW']['RMSE']
avg_normalized_mae_hw = normalized_metrics['HW']['MAE']
avg_normalized_mape_hw = normalized_metrics['HW']['MAPE']
avg_normalized_mase_hw = normalized_metrics['HW']['MASE']

avg_normalized_rmse_arima = normalized_metrics['ARIMA']['RMSE']
avg_normalized_mae_arima = normalized_metrics['ARIMA']['MAE']
avg_normalized_mape_arima = normalized_metrics['ARIMA']['MAPE']
avg_normalized_mase_arima = normalized_metrics['ARIMA']['MASE']

avg_normalized_rmse_sarima = normalized_metrics['SARIMA']['RMSE']
avg_normalized_mae_sarima = normalized_metrics['SARIMA']['MAE']
avg_normalized_mape_sarima = normalized_metrics['SARIMA']['MAPE']
avg_normalized_mase_sarima = normalized_metrics['SARIMA']['MASE']

avg_normalized_rmse_prophet = normalized_metrics['Prophet']['RMSE']
avg_normalized_mae_prophet = normalized_metrics['Prophet']['MAE']
avg_normalized_mape_prophet = normalized_metrics['Prophet']['MAPE']
avg_normalized_mase_prophet = normalized_metrics['Prophet']['MASE']
```

Decomposizione STL

```

```

```
# Eseguire la decomposizione STL
stl = STL(df_subset_0817, seasonal=181)  # Seasonal_period puo essere cambiato in base alla stagionalita' rilevata
result = stl.fit()

# Ottenere le componenti
trend = result.trend
seasonal = result.seasonal
residual = result.resid

# Calcolare le varianze
var_residual = np.var(residual)
var_trend_plus_residual = np.var(trend + residual)
var_seasonal_plus_residual = np.var(seasonal + residual)

# Calcolare la forza del trend
strength_trend = max(0, 1 - (var_residual / var_trend_plus_residual))

# Calcolare la forza della stagionalita'
strength_seasonal = max(0, 1 - (var_residual / var_seasonal_plus_residual))

# Stampa dei risultati
print(f"Forza del Trend: {strength_trend}")
print(f"Forza della Stagionalita': {strength_seasonal}")
```

Calcolo della metrica CFEM

```

# Pesi componente w
w_RMSE = 0.25
w_MAE = 0.3
w_MAPE = 0.2
w_MASE = 0.25

# Pesi metriche in base al modello
# Holt Winters
z_RMSE_hw = 1 / med_rmse_hw
z_MAE_hw = 1 / med_mae_hw
z_MAPE_hw = 1 / med_mape_hw
z_MASE_hw = 1 / med_mase_hw
z_sum_hw = z_RMSE_hw + z_MAE_hw + z_MAPE_hw + z_MASE_hw

# Prophet
z_RMSE_prophet = 1 / med_rmse_prophet
z_MAE_prophet = 1 / med_mae_prophet
z_MAPE_prophet = 1 / med_mape_prophet
z_MASE_prophet = 1 / med_mase_prophet
z_sum_prophet = z_RMSE_prophet + z_MAE_prophet + z_MAPE_prophet + z_MASE_prophet

# ARIMA
z_RMSE_arima = 1 / med_rmse_arima
z_MAE_arima = 1 / med_mae_arima
z_MAPE_arima = 1 / med_mape_arima
z_MASE_arima = 1 / med_mase_arima
```

```
z_sum_arima = z_RMSE_arima + z_MAE_arima + z_MAPE_arima +
    z_MASE_arima

# SARIMA
z_RMSE_sarima = 1 / med_rmse_sarima
z_MAE_sarima = 1 / med_mae_sarima
z_MAPE_sarima = 1 / med_mape_sarima
z_MASE_sarima = 1 / med_mase_sarima
z_sum_sarima = z_RMSE_sarima + z_MAE_sarima + z_MAPE_sarima +
    z_MASE_sarima

# Stampa Holt Winters
print("HOLT WINTERS")

RMSE_weight_hw = ((strength_seasonal) * z_RMSE_hw) * w_RMSE
MAE_weight_hw = ((strength_seasonal) * z_MAE_hw) * w_MAE
MAPE_weight_hw = ((strength_seasonal) * z_MAPE_hw) * w_MAPE
MASE_weight_hw = ((strength_seasonal) * z_MASE_hw) * w_MASE

print(" ")
cfem_hw = (RMSE_weight_hw * avg_normalized_rmse_hw + MAE_weight_hw
    * avg_normalized_mae_hw +
MAPE_weight_hw * avg_normalized_mape_hw + MASE_weight_hw *
    avg_normalized_mase_hw)
print(f"CFEM per modello Holt-Winters: {cfem_hw / z_sum_hw}")

print("\n\n")

# Stampa Prophet
print("PROPHET")

RMSE_weight_prophet = ((strength_seasonal) * z_RMSE_prophet) *
    w_RMSE
MAE_weight_prophet = ((strength_seasonal) * z_MAE_prophet) * w_MAE
MAPE_weight_prophet = ((strength_seasonal) * z_MAPE_prophet) *
    w_MAPE
MASE_weight_prophet = ((strength_seasonal) * z_MASE_prophet) *
    w_MASE

print(f"CFEM per modello Prophet: {(RMSE_weight_prophet *
    avg_normalized_rmse_prophet + MAE_weight_prophet *
    avg_normalized_mae_prophet + MAPE_weight_prophet *
    avg_normalized_mape_prophet + MASE_weight_prophet *
    avg_normalized_mase_prophet) / z_sum_prophet}")

print("\n\n")

# Stampa ARIMA
print("ARIMA")

RMSE_weight_arima = ((strength_seasonal) * z_RMSE_arima) * w_RMSE
MAE_weight_arima = ((strength_seasonal) * z_MAE_arima) * w_MAE
MAPE_weight_arima = ((strength_seasonal) * z_MAPE_arima) * w_MAPE
MASE_weight_arima = ((strength_seasonal) * z_MASE_arima) * w_MASE

print(f"CFEM per modello ARIMA: {(RMSE_weight_arima *
    avg_normalized_rmse_arima + MAE_weight_arima *
    avg_normalized_mae_arima +MAPE_weight_arima *
```

```
    avg_normalized_mape_arima + MASE_weight_arima * avg_normalized_mase_arima) / z_sum_arima}")

print("\n\n")

# Stampa SARIMA
print("SARIMA")

RMSE_weight_sarima =  ((strength_seasonal) * z_RMSE_sarima) * w_RMSE
MAE_weight_sarima =  ((strength_seasonal) * z_MAE_sarima) * w_MAE
MAPE_weight_sarima = ((strength_seasonal) * z_MAPE_sarima) * w_MAPE
MASE_weight_sarima =  ((strength_seasonal) * z_MASE_sarima) * w_MASE

print(f"CFEM per modello SARIMA: {(RMSE_weight_sarima * avg_normalized_rmse_sarima + MAE_weight_sarima * avg_normalized_mae_sarima + MAPE_weight_sarima * avg_normalized_mape_sarima + MASE_weight_sarima * avg_normalized_mase_sarima) / z_sum_sarima}")
```

A.0.2 Dataset aziendale

Setup Dataset e preprocessing

```

# Import delle librerie utili
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_absolute_error, mean_squared_error, root_mean_squared_error


from google.colab import files
uploaded = files.upload()
df = pd.read_csv('dataset_azienda.csv')
df.head()


# Converti la colonna Datetime in DateTimeIndex e ordina i dati
df['Date'] = pd.to_datetime(df['Date'])
df_sorted = df.sort_values(by='Date').reset_index(drop=True)

# Seleziona le colonne di interesse
df_timeseries_3 = df_sorted[['Date', '3']]

# Plot della serie temporale
plt.figure(figsize=(12, 6))
plt.plot(df_sorted['Date'], df_sorted['3'], linewidth=3, c='blue')
plt.title("Timeseries 3")
plt.xlabel("Date")
plt.ylabel("3")
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()

# Scatter plot della serie temporale
plt.figure(figsize=(12, 6))
plt.scatter(df_sorted['Date'], df_sorted['3'], c='blue', marker='o', alpha=0.5)
plt.xlabel("Data")
plt.ylabel("Vendite")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Prepara i dati per Prophet
df_prophet = df_timeseries_3.copy()

# Imposta la colonna 'Date' come indice
df_timeseries_3.set_index('Date', inplace=True)

# Assicurati che l'indice sia un DatetimeIndex
df_timeseries_3.index = pd.to_datetime(df_timeseries_3.index, errors='coerce')

# Imposta la frequenza dell'indice
df_timeseries_3 = df_timeseries_3.asfreq('D')
print(f"Frequenza dell'indice: {df_timeseries_3.index.freq}")

# Assegna la colonna '3' alla variabile y
y = df_timeseries_3['3']

# Crea una copia del dataset originale utile per il modello Prophet
df_prophet_ts3 = df_prophet
# Rinomina le colonne per Prophet
df_prophet_ts3 = df_prophet_ts3.rename(columns={'Date': 'ds', '3': 'y'})

# Verifica che la rinomina sia avvenuta correttamente
print("Colonne dopo la rinomina per Prophet:", df_prophet_ts3.columns.tolist())

# Verifica il risultato finale
print(df_prophet_ts3.head())
```

```

# Gestione degli outliers

import pandas as pd

# Filtra solo i dati del 2023 usando l'indice invece della colonna 'Date' per df_timeseries_3
df_2023 = df_timeseries_3[df_timeseries_3.index.year == 2023].copy()

# Calcolo dei quartili e dell'Interquartile Range (IQR)
Q1 = df_2023['3'].quantile(0.25)
Q3 = df_2023['3'].quantile(0.75)
IQR = Q3 - Q1

```

```
# Definizione dei limiti accettabili
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Calcolo della mediana dei valori accettabili
median_value = df_2023[(df_2023['3'] >= lower_bound) & (df_2023['3'] <= upper_bound)]['3'].median()

# Sostituisce gli outliers con la mediana
df_2023.loc[(df_2023['3'] < lower_bound) | (df_2023['3'] > upper_bound), '3'] = median_value

# Unisce i dati modificati del 2023 con il resto del dataset
df_cleaned = pd.concat([df_timeseries_3[df_timeseries_3.index.year != 2023], df_2023])

# Ordina il dataframe per data
df_cleaned = df_cleaned.sort_index()

# Mostra il numero di outliers sostituiti
print(f"Outliers sostituiti in df_timeseries_3: {(df_2023['3'] < lower_bound).sum() + (df_2023['3'] > upper_bound).sum()}")

# ------ PULIZIA DEL DATAFRAME PER PROPHET ------

# Filtra solo i dati del 2023 per df_prophet_ts3
df_2023_prophet = df_prophet_ts3[df_prophet_ts3['ds'].dt.year == 2023].copy()

# Calcolo dei quartili e dell'Interquartile Range (IQR)
Q1_p = df_2023_prophet['y'].quantile(0.25)
Q3_p = df_2023_prophet['y'].quantile(0.75)
IQR_p = Q3_p - Q1_p

# Definizione dei limiti accettabili
lower_bound_p = Q1_p - 1.5 * IQR_p
upper_bound_p = Q3_p + 1.5 * IQR_p

# Calcolo della mediana dei valori accettabili
median_value_p = df_2023_prophet[(df_2023_prophet['y'] >= lower_bound_p) & (df_2023_prophet['y'] <= upper_bound_p)]['y'].median()

# Sostituisce gli outliers con la mediana
df_2023_prophet['y'] = df_2023_prophet['y'].apply(lambda x: median_value_p if x < lower_bound_p or x > upper_bound_p else x)

# Unisce i dati modificati del 2023 con il resto del dataset
df_prophet_ts3_cleaned = pd.concat([df_prophet_ts3[df_prophet_ts3['ds'].dt.year != 2023], df_2023_prophet])

# Ordina il dataframe per data
df_prophet_ts3_cleaned = df_prophet_ts3_cleaned.sort_values(by='ds')

# Mostra quanti outliers sono stati sostituiti
outliers_count_p = sum((df_2023_prophet['y'] < lower_bound_p) | (df_2023_prophet['y'] > upper_bound_p))
```

```
print(f"Outliers sostituiti in df_prophet_ts3: {outliers_count_p}")

y = df_cleaned['3']
```

```
        pip install prophet --no-binary :all:
```

```
        !pip install prophet
```

```
        pip install pmdarima
```

Cross-Validation e modelli

```

import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tools.sm_exceptions import ConvergenceWarning
from sklearn.metrics import mean_absolute_error
from prophet import Prophet

warnings.simplefilter('ignore', ConvergenceWarning)

def root_mean_squared_error(y_true, y_pred):
return np.sqrt(np.mean((y_true - y_pred) ** 2))

# Impostazioni per la cross-validation
n_test_years = 365
n_splits = len(y) // n_test_years

# Liste per raccogliere le metriche
metrics = {'HW': [], 'ARIMA': [], 'SARIMA': [], 'Prophet': []}

for i in range(n_splits - 1):
train_end = (i + 1) * n_test_years
train, test = y[:train_end], y[train_end:train_end + n_test_years]
train_prophet, test_prophet = df_prophet_ts3_cleaned.iloc[:
    train_end], df_prophet_ts3_cleaned.iloc[train_end:train_end +
    n_test_years]

models = {
        'HW': ExponentialSmoothing(train, trend='add', seasonal='
            add', seasonal_periods=60).fit(optimized=True),
        'ARIMA': ARIMA(train, order=(5, 1, 5)).fit(),
        'SARIMA': SARIMAX(train, order=(5, 1, 5), seasonal_order
            =(1, 1, 1, 36)).fit()
}

for model_name, model in models.items():
forecast = model.forecast(len(test))
rmse = root_mean_squared_error(test, forecast)
mae = mean_absolute_error(test, forecast)
mape = (abs(test - forecast) / test).mean() * 100
```

```
mase = mae / train.mean()
metrics[model_name].append((rmse, mae, mape, mase))
print(f"{model_name} - Fold {i+1}: RMSE={rmse:.2f}, MAE={mae:.2f}, MAPE={mape:.2f}, MASE={mase:.4f}")
print("-" * 30)

# Prophet - Nuovo modello a ogni iterazione
prophet_model = Prophet(yearly_seasonality=True, daily_seasonality=False)
prophet_model.fit(train_prophet)

future = prophet_model.make_future_dataframe(periods=len(test), freq='D')
forecast_prophet = prophet_model.predict(future)['yhat'].iloc[-len(test):].values

rmse = root_mean_squared_error(test_prophet['y'].values, forecast_prophet)
mae = mean_absolute_error(test_prophet['y'].values, forecast_prophet)
mape = (abs(test_prophet['y'].values - forecast_prophet) / test_prophet['y'].values).mean() * 100
mase = mae / train_prophet['y'].mean()
metrics['Prophet'].append((rmse, mae, mape, mase))
print(f"Prophet - Fold {i+1}: RMSE={rmse:.2f}, MAE={mae:.2f}, MAPE={mape:.2f}, MASE={mase:.4f}")
print("-" * 30)

# Grafico separato per ogni modello
for model_name, model in models.items():
plt.figure(figsize=(12, 6))
plt.plot(y.index, y, label='Original Data', color='blue')
plt.plot(test.index, model.forecast(len(test)), label=f'{model_name} Forecast', color='red')
plt.xlabel('Date'); plt.ylabel('AEP_MW')
plt.title(f'Forecast Last Fold - {model_name}')
plt.legend(); plt.xticks(rotation=45); plt.tight_layout()
plt.show()

# Prophet forecast plot separato (nuova istanza)
prophet_model = Prophet(yearly_seasonality=True, daily_seasonality=False)
prophet_model.fit(df_prophet_ts3_cleaned.iloc[:train_end])
future = prophet_model.make_future_dataframe(periods=len(test), freq='D')
pred_prophet = prophet_model.predict(future)['yhat'].iloc[-len(test):].values

plt.figure(figsize=(12, 6))
plt.plot(y.index, y, label='Original Data', color='blue')
plt.plot(test.index, pred_prophet, label='Prophet Forecast', color='red')
plt.xlabel('Date'); plt.ylabel('AEP_MW')
plt.title('Forecast Last Fold - Prophet')
plt.legend(); plt.xticks(rotation=45); plt.tight_layout()
plt.show()

# Stampa medie e mediane delle metriche
```

```
for model_name, values in metrics.items():
values = np.array(values)
print("␣")
print(f"{model_name}␣-␣Media:␣RMSE={values[:,0].mean():.2f},␣MAE={
    values[:,1].mean():.2f},␣MAPE={values[:,2].mean():.2f},␣MASE={
    values[:,3].mean():.4f}")
print(f"{model_name}␣-␣Mediana:␣RMSE={np.median(values[:,0]):.2f},␣
    MAE={np.median(values[:,1]):.2f},␣MAPE={np.median(values[:,2])
    :.2f},␣MASE={np.median(values[:,3]):.4f}\n")
print("␣")

# Estrarre le mediane delle metriche per ciascun modello
med_rmse_hw = np.median(np.array(metrics['HW'])[:, 0])
med_mae_hw = np.median(np.array(metrics['HW'])[:, 1])
med_mape_hw = np.median(np.array(metrics['HW'])[:, 2])
med_mase_hw = np.median(np.array(metrics['HW'])[:, 3])

med_rmse_arima = np.median(np.array(metrics['ARIMA'])[:, 0])
med_mae_arima = np.median(np.array(metrics['ARIMA'])[:, 1])
med_mape_arima = np.median(np.array(metrics['ARIMA'])[:, 2])
med_mase_arima = np.median(np.array(metrics['ARIMA'])[:, 3])

med_rmse_sarima = np.median(np.array(metrics['SARIMA'])[:, 0])
med_mae_sarima = np.median(np.array(metrics['SARIMA'])[:, 1])
med_mape_sarima = np.median(np.array(metrics['SARIMA'])[:, 2])
med_mase_sarima = np.median(np.array(metrics['SARIMA'])[:, 3])

med_rmse_prophet = np.median(np.array(metrics['Prophet'])[:, 0])
med_mae_prophet = np.median(np.array(metrics['Prophet'])[:, 1])
med_mape_prophet = np.median(np.array(metrics['Prophet'])[:, 2])
med_mase_prophet = np.median(np.array(metrics['Prophet'])[:, 3])
```

Normalizzazione min-max delle metriche ottenute

```
# Funzione di normalizzazione Min-Max
def min_max_normalization(value, min_value, max_value):
return (value - min_value) / (max_value - min_value) if max_value
    != min_value else 0

# Dizionario per raccogliere le metriche normalizzate
normalized_metrics = {}

for model_name in metrics.keys():
values = np.array(metrics[model_name])

min_rmse, max_rmse = values[:, 0].min(), values[:, 0].max()
min_mae, max_mae = values[:, 1].min(), values[:, 1].max()
min_mape, max_mape = values[:, 2].min(), values[:, 2].max()
min_mase, max_mase = values[:, 3].min(), values[:, 3].max()

normalized_rmse = [min_max_normalization(rmse, min_rmse, max_rmse)
    for rmse in values[:, 0]]
normalized_mae = [min_max_normalization(mae, min_mae, max_mae) for
    mae in values[:, 1]]
normalized_mape = [min_max_normalization(mape, min_mape, max_mape)
    for mape in values[:, 2]]
```

```
normalized_mase = [min_max_normalization(mase, min_mase, max_mase)
    for mase in values[:, 3]]

avg_normalized_rmse = np.mean(normalized_rmse)
avg_normalized_mae = np.mean(normalized_mae)
avg_normalized_mape = np.mean(normalized_mape)
avg_normalized_mase = np.mean(normalized_mase)

# Salva nel dizionario
normalized_metrics[model_name] = {
        'RMSE': avg_normalized_rmse,
        'MAE': avg_normalized_mae,
        'MAPE': avg_normalized_mape,
        'MASE': avg_normalized_mase
}

print(f"Media delle metriche normalizzate di tutti i fold ({
    model_name}):")
print(f"RMSE (normalizzato): {avg_normalized_rmse}")
print(f"MAE (normalizzato): {avg_normalized_mae}")
print(f"MAPE (normalizzato): {avg_normalized_mape}")
print(f"MASE (normalizzato): {avg_normalized_mase}")
print("-" * 60)
print(" ")

# Estrai i valori dal dizionario e assegna alle variabili
    necessarie
avg_normalized_rmse_hw = normalized_metrics['HW']['RMSE']
avg_normalized_mae_hw = normalized_metrics['HW']['MAE']
avg_normalized_mape_hw = normalized_metrics['HW']['MAPE']
avg_normalized_mase_hw = normalized_metrics['HW']['MASE']

avg_normalized_rmse_arima = normalized_metrics['ARIMA']['RMSE']
avg_normalized_mae_arima = normalized_metrics['ARIMA']['MAE']
avg_normalized_mape_arima = normalized_metrics['ARIMA']['MAPE']
avg_normalized_mase_arima = normalized_metrics['ARIMA']['MASE']

avg_normalized_rmse_sarima = normalized_metrics['SARIMA']['RMSE']
avg_normalized_mae_sarima = normalized_metrics['SARIMA']['MAE']
avg_normalized_mape_sarima = normalized_metrics['SARIMA']['MAPE']
avg_normalized_mase_sarima = normalized_metrics['SARIMA']['MASE']

avg_normalized_rmse_prophet = normalized_metrics['Prophet']['RMSE']
avg_normalized_mae_prophet = normalized_metrics['Prophet']['MAE']
avg_normalized_mape_prophet = normalized_metrics['Prophet']['MAPE']
avg_normalized_mase_prophet = normalized_metrics['Prophet']['MASE']
```

Decomposizione STL

```
import numpy as np
import pandas as pd
from statsmodels.tsa.seasonal import STL


# Eseguire la decomposizione STL
stl = STL(df_cleaned, seasonal=185)  # Seasonal_period puo' essere
    cambiato in base alla stagionalita' rilevata
```

```
result = stl.fit()

# Ottenere le componenti
trend = result.trend
seasonal = result.seasonal
residual = result.resid

# Calcolare le varianze
var_residual = np.var(residual)
var_trend_plus_residual = np.var(trend + residual)
var_seasonal_plus_residual = np.var(seasonal + residual)

# Calcolare la forza del trend
strength_trend = max(0, 1 - (var_residual / var_trend_plus_residual
    ))

# Calcolare la forza della stagionalita'
strength_seasonal = max(0, 1 - (var_residual /
    var_seasonal_plus_residual))

# Stampa dei risultati
print(f"Forza del Trend: {strength_trend}")
print(f"Forza della Stagionalita': {strength_seasonal}")
```

Calcolo della metrica CFEM

```

# Pesi componente w
w_RMSE = 0.2
w_MAE = 0.3
w_MAPE = 0
w_MASE = 0.5

# Controllo per escludere MAPE se il peso e' 0
if w_MAPE == 0:
avg_normalized_mape = 0
avg_normalized_mape_arima = 0
avg_normalized_mape_prophet = 0
avg_normalized_mape_sarima = 0

# Pesi metriche in base al modello
# Holt-Winters
m_RMSE_hw = 1 / med_rmse_hw
m_MAE_hw = 1 / med_mae_hw
m_MAPE_hw = 1 / med_mape_hw
m_MASE_hw = 1 / med_mase_hw
m_sum_hw = m_RMSE_hw + m_MAE_hw + (m_MAPE_hw if w_MAPE > 0 else 0)
    + m_MASE_hw

# Prophet
m_RMSE_prophet = 1 / med_rmse_prophet
m_MAE_prophet = 1 / med_mae_prophet
m_MAPE_prophet = 1 / med_mape_prophet
m_MASE_prophet = 1 / med_mase_prophet
m_sum_prophet = m_RMSE_prophet + m_MAE_prophet + (m_MAPE_prophet if
     w_MAPE > 0 else 0) + m_MASE_prophet

```

```
# ARIMA
m_RMSE_arima = 1 / med_rmse_arima
m_MAE_arima = 1 / med_mae_arima
m_MAPE_arima = 1 / med_mape_arima
m_MASE_arima = 1 / med_mase_arima
m_sum_arima = m_RMSE_arima + m_MAE_arima + (m_MAPE_arima if w_MAPE
    > 0 else 0) + m_MASE_arima

# SARIMA
m_RMSE_sarima = 1 / med_rmse_sarima
m_MAE_sarima = 1 / med_mae_sarima
m_MAPE_sarima = 1 / med_mape_sarima
m_MASE_sarima = 1 / med_mase_sarima
m_sum_sarima = m_RMSE_sarima + m_MAE_sarima + (m_MAPE_sarima if
    w_MAPE > 0 else 0) + m_MASE_sarima

# Stampa Holt-Winters
print("HOLT-WINTERS")

RMSE_weight_hw = ((strength_seasonal) * m_RMSE_hw) * w_RMSE
MAE_weight_hw = ((strength_seasonal) * m_MAE_hw) * w_MAE
MAPE_weight_hw = ((strength_seasonal) * m_MAPE_hw) * w_MAPE
MASE_weight_hw = ((strength_seasonal) * m_MASE_hw) * w_MASE

print(f"CFEM␣per␣modello␣Holt-Winters:␣{(RMSE_weight_hw␣*␣
    avg_normalized_rmse␣+␣MAE_weight_hw␣*␣avg_normalized_mae␣+␣
    MAPE_weight_hw␣*␣avg_normalized_mape␣+␣MASE_weight_hw␣*␣
    avg_normalized_mase)␣/␣m_sum_hw}")

print("\n\n")

# Stampa Prophet
print("PROPHET")

RMSE_weight_prophet = ((strength_seasonal) * m_RMSE_prophet) *
    w_RMSE
MAE_weight_prophet = ((strength_seasonal) * m_MAE_prophet) * w_MAE
MAPE_weight_prophet = ((strength_seasonal) * m_MAPE_prophet) *
    w_MAPE
MASE_weight_prophet = ((strength_seasonal) * m_MASE_prophet) *
    w_MASE

print(f"CFEM␣per␣modello␣Prophet:␣{(RMSE_weight_prophet␣*␣
    avg_normalized_rmse_prophet␣+␣MAE_weight_prophet␣*␣
    avg_normalized_mae_prophet␣+␣MAPE_weight_prophet␣*␣
    avg_normalized_mape_prophet␣+␣MASE_weight_prophet␣*␣
    avg_normalized_mase_prophet)␣/␣m_sum_prophet}")

print("\n\n")

# Stampa ARIMA
print("ARIMA")

RMSE_weight_arima = ((strength_seasonal) * m_RMSE_arima) * w_RMSE
MAE_weight_arima = ((strength_seasonal) * m_MAE_arima) * w_MAE
MAPE_weight_arima = ((strength_seasonal) * m_MAPE_arima) * w_MAPE
MASE_weight_arima = ((strength_seasonal) * m_MASE_arima) * w_MASE

```

```
print(f"CFEM␣per␣modello␣ARIMA:␣{(RMSE_weight_arima␣*␣
    avg_normalized_rmse_arima␣+␣MAE_weight_arima␣*␣
    avg_normalized_mae_arima␣+␣MAPE_weight_arima␣*␣
    avg_normalized_mape_arima␣+␣MASE_weight_arima␣*␣
    avg_normalized_mase_arima)␣/␣m_sum_arima}")

print("\n\n")

# Stampa SARIMA
print("SARIMA")

RMSE_weight_sarima = ((strength_seasonal) * m_RMSE_sarima) * w_RMSE
MAE_weight_sarima = ((strength_seasonal) * m_MAE_sarima) * w_MAE
MAPE_weight_sarima = ((strength_seasonal) * m_MAPE_sarima) * w_MAPE
MASE_weight_sarima = ((strength_seasonal) * m_MASE_sarima) * w_MASE

print(f"CFEM␣per␣modello␣SARIMA:␣{(RMSE_weight_sarima␣*␣
    avg_normalized_rmse_sarima␣+␣MAE_weight_sarima␣*␣
    avg_normalized_mae_sarima␣+␣MAPE_weight_sarima␣*␣
    avg_normalized_mape_sarima␣+␣MASE_weight_sarima␣*␣
    avg_normalized_mase_sarima)␣/␣m_sum_sarima}")
```

Bibliografia

Alonso, Gustavo et al. (giu. 2021). "Comparison between SARIMA and Holt–Winters models for forecasting monthly streamflow in the western region of Cuba". In: *SN Applied Sciences* 3. DOI: 10.1007/s42452-021-04667-5.

Ersoz, N.S. et al. (2022). "Comparative Performance Analysis of ARIMA, Prophet and Holt-Winters Forecasting Methods on European COVID-19 Data". In: *International Journal of 3D Printing Technologies and Digital Industry* 6.3, pp. 556–565. DOI: 10.46519/ij3dptdi.1120718. URL: https://dergipark.org.tr/tr/download/article-file/2444859.

Galdino, Eraylson (2023). *The main patterns in Time Series*. URL: https://medium.com/@eraylson_/the-main-patterns-in-time-series-7a582054ef9a#:~:text=A%20linear%20trend%20model%20is,is%20not%20constant%20over%20time..

Hayes, Adam (2024). *Autoregressive Integrated Moving Average (ARIMA) Prediction Model*. URL: https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp.

Hodson, T. O. (giu. 2022). "Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not". In: *Geoscientific Model Development* 15.14, pp. 5481–5487. DOI: 10.5194/gmd-15-5481-2022. URL: https://gmd.copernicus.org/articles/15/5481/2022/.

Hyndman, Rob J. e George Athanasopoulos (2018). *Forecasting: Principles and Practice*. OTexts. URL: https://otexts.com/fpp2/.

Kis, Andras Nandor (2024). *Demystifying STL: Understanding Seasonal Decomposition of Time Series*. URL: https://medium.com/@kis.andras.nandor/demystifying-stl-understanding-seasonal-decomposition-of-time-series-d3c50150ec12.

Kwarteng, Samuel Baffoe e Poguda Aleksey Andreevich (2024). "Comparative Analysis of ARIMA, SARIMA and Prophet Model in Forecasting". In: *Research and Development* 5.4, pp. 110–120. DOI: 10.11648/j.rd.20240504.13. eprint: https://article.sciencepublishinggroup.com/pdf/10.11648.j.rd.20240504.13. URL: https://doi.org/10.11648/j.rd.20240504.13.

Maklin, Cory (2024). *Fast Fourier Transform Explained*. URL: https://builtin.com/articles/fast-fourier-transform.

Palvel, Subash (2023). *Understanding Time Series Cross-validation*. URL: https://subashpalvel.medium.com/understanding-time-series-cross-validation-1929c543d339.

Tejalk, Adam (2024). *What is a trend in time series?* URL: https://www.geeksforgeeks.org/what-is-a-trend-in-time-series/.

Vien, Benjamin et al. (feb. 2021). "A Machine Learning Approach for Anaerobic Reactor Performance Prediction Using Long Short-Term Memory Recurrent Neural Network". In: DOI: 10.21741/9781644901311-8.

Wang, X., K. Smith e R. Hyndman (2006). "Characteristic-Based Clustering for Time Series Data". In: *Data Mining and Knowledge Discovery* 13, pp. 335–364. DOI: https://doi.org/10.1007/s10618-005-0039-x.

Xia, Genglei, Yuepeng Bi e Chenyang Wang (2023). "Optimization design of passive residual heat removal system based on improved genetic algorithm". In: *Annals of Nuclear Energy* 189, p. 109859. ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2023.109859. URL: https://www.sciencedirect.com/science/article/pii/S0306454923001780.