UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Analisi delle risorse idriche e ottimizzazione energetica dei pozzi della sorgente Gorgovivo con tecniche data-driven basate su Machine Learning

Analysis of water resources and energy optimization of wells in the Gorgovivo spring based on data-driven Machine Learning techniques

Relatore:
Dr. Galdelli Alessandro

Laureando:
Coppari Alessio

Correlatore:
Prof. Zingaretti Primo

Anno Accademico 2025-2026

Indice

Elenco delle figure **5**

Sommario **6**

Summary **7**

1 Introduzione **8**
1.1 Motivazioni e Obiettivi 11
1.2 Struttura Tesi 12

2 La sorgente Gorgovivo **13**
2.1 Storia 13
2.2 Descrizione sito 13

3 Previsione di Serie Temporali **15**
3.1 Serie Temporali 15
3.1.1 Pre-processamento 17
3.1.2 Analisi Dati Esplorativa 20
3.1.3 Correlazione 20
3.2 Previsione 24
3.2.1 Applicazioni 24
3.2.2 Previsione di Serie Temporali Multivariate 25
3.2.3 Metodi Statistici 26
3.2.4 Metodi di Machine Learning 28
3.2.5 Deep Learning 30
3.3 Lavori Correlati 32

4 Materiali e Metodi **34**
4.1 Ambiente di sviluppo 34
4.1.1 Python 34
4.1.2 Visual Studio Code 35
4.1.3 Google Colab 35
4.1.4 Microsoft Excel 36
4.1.5 Modelli di Intelligenza Artificiale 36

4.2 SCADA 36
4.2.1 File di Input 36
4.3 Metodologia per l'ottimizzazione energetica 38

5 Progettazione e Sviluppo 41
5.1 Raccolta dei dati 42
5.1.1 Lettura file 42
5.1.2 Costruzione Dataset 46
5.2 Analisi dei dati 50
5.2.1 Gestione dei dati mancanti 50
5.2.2 Analisi Dati Esplorativa 56
5.3 Ottimizzazione 80
5.3.1 Test di ottimizzazione 80
5.3.2 Risultati 87
5.4 Benchmark 90

6 Discussioni e Conclusioni 94
6.1 Sviluppi futuri 95

Bibliografia 97

Elenco delle figure

1.1 Distribuzione dell'acqua nel mondo [1]. 9
1.2 Consumo energetico nell'UE [2]. 10

3.1 Decomposizione di serie di dati numerici sintetici [3]. 16
3.2 Finestra di Sakoe-Chiba [4]. 22
3.3 Filtro di Hampel [5]. 23
3.4 Parole chiave su applicazioni di ML per risorse idriche [6]. 32

4.1 Esempio di file input. 37
4.2 Metodologia per l' ottimizzazione energetica. 38
4.3 Struttura del problema di ottimizzazione: obiettivi e vincoli operativi. 39

5.1 Porzione del report sulle variabili. 51
5.2 Percentuale NaN per ogni parametro. 52
5.3 Dati validi e mancanti per la Portata del Pozzo 14B. 53
5.4 Dati validi e mancanti per la Portata della Pompa 1 del Pozzo 10. 53
5.5 Dati validi e mancanti per la Portata della Pompa 2 del Pozzo 9. . 53
5.6 Dati validi e mancanti per la Portata della Pompa 1 del Pozzo 9. . 54
5.7 Porzione del report sulle variabili aggiornato. 55
5.8 Portata e Corrente delle Pompe del Pozzo 12. 58
5.9 Portata e Corrente delle Pompe del Pozzo 6. 58
5.10 Portata e Corrente del Pozzo 14B. 58
5.11 Porzione del report sulle distanze di DTW per Portata-Corrente. . 60
5.12 Correlazione Portata-Corrente per ciascuna Pompa di ogni Pozzo. 61
5.13 Contributo di ogni Pompa alla Portata complessiva di ciascun Pozzo. 63
5.14 Contributo di ogni Pompa alla Corrente complessiva consumata dalle pompe di ciascun Pozzo. 64
5.15 Contributo di ogni Pozzo alla Portata totale dell'intero sistema. . . 66
5.16 Contributo di ogni Pozzo al consumo di Corrente totale dell'intero sistema. 66
5.17 Livelli d'acqua nei pozzi mensili medi calcolati su tutti gli anni. . . 68
5.18 Livello-Portata Pozzo 1. 70

5.19 Porzione del report sulle distanze di DTW per Livello-Portata. . . 70
5.20 Livello Serbatoio. 72
5.21 Portata in Uscita Serbatoio. 72
5.22 Livello e Portata in Uscita (Normalizzati). 72
5.23 Correlazione Livello-Portata in Uscita (Serbatoio). 73
5.24 Distanze di DTW per Livello-Portata in Uscita (Serbatoio). 73
5.25 Andamento medio giornaliero della Portata in Uscita e del Livello d'acqua del Serbatoio. 75
5.26 Autocorrelazione (Serbatoio). 76
5.27 Dati dei Pozzi Aggregati. 77
5.28 Correlazione Portata in Entrata-Livello. 78
5.29 Distanze di DTW per Portata in Entrata-Livello (Serbatoio). . . . 78
5.30 Grafo dei Pozzi. 82
5.31 Previsione richiesta idrica. 88
5.32 Previsione Livelli nei Pozzi. 88
5.33 Andamento del Livello del Serbatoio durante l'ottimizzazione. . . . 88
5.34 Correnti ottime allocate ad ogni pozzo. 88
5.35 Portata in Ingresso vs. Domanda. 89
5.36 Tempo di utilizzo Pompe. 89
5.37 Corrente consumata manualmente vs. ottimizzata. 89
5.38 Benchmark Pozzi. 92
5.39 Benchmark Domanda. 93
5.40 Metriche di Valutazione. 93

Sommario

La salvaguardia delle risorse idriche e la sostenibilità ecologica sono degli aspetti critici divenuti imprescindibili in un contesto di crisi climatica e di scarsità d'acqua. L'aumento della richiesta idrica e del consumo di energia elettrica è un problema che incide lo sviluppo di società di tutto il mondo. Tra i metodi innovativi che vengono ricercati in questo scenario, l'intelligenza artificiale trova un ruolo di rilievo. Questa tecnologia, oggi applicata in innumerevoli ambiti, può dare risultati promettenti nella gestione delle risorse idriche e nella riduzione del consumo energetico che essa comporta.
La presente tesi si propone di applicare modelli di intelligenza artificiale, nello specifico di machine learning, nell'ottimizzazione della gestione dell'acquedotto della sorgente Gorgovivo, situata a Serra San Quirico.
Questo progetto è stato svolto in collaborazione con il Consorzio Gorgovivo e VIVA Servizi S.p.A. con lo scopo di creare un sistema in grado di ottimizzare la gestione della sorgente, minimizzando i consumi di energia elettrica e riducendo gli sprechi d'acqua. Per raggiungere questo obiettivo è stata svolta un'analisi approfondita dei dati reali del sistema e, in base alle informazioni ottenute, si è sviluppata una simulazione di ottimizzazione che ha come nucleo dei modelli di machine learning per effettuare previsioni sullo stato futuro del sistema, che determinano il criterio ottimale secondo cui vengono attivati i pozzi della sorgente.
Questo lavoro rappresenta una base sulla quale si può costruire un programma completo che, in seguito ai dovuti test, potrà essere integrato ai sistemi reali dell'acquedotto come strumento di supporto decisionale.

Summary

Safekeeping of water resources and ecological sustainability are critical aspects that have become essential in the context of climate crisis and water scarcity. The increase in water demand and electricity consumption has become a problem that hinders the development of societies around the world. Among the innovative methods researched in this scenario, artificial intelligence plays a key role. This technology, currently applied countless fields, can produce promising results in water management and the reduction of energy consumption it entails. This thesis aims to apply artificial intelligence models, specifically machine learning, to optimize the management of the aqueduct of the Gorgovivo spring, located in Serra San Quirico.
This project has been developed in a collaboration with Consorzio Gorgovivo and VIVA Serivizi S.p.A., aiming to create a system capable of optimizing spring management, minimizing electricity consumption and reducing water waste. To reach this goal, an in-depth analysis of the system's real data has been carried out and, based on the newly obtained information, an optimization simulation has been developed. This simulation is built with machine learning models at its core that make forecasts on the system's future state, which determine the optimal criteria to activate the spring wells.
This work provides a foundation on which a comprehensive program can be built which, following appropriate testing, can be integrated into the aqueduct's actual systems as a decision-making support tool.

Capitolo 1

Introduzione

L'acqua è la risorsa naturale più importante per la vita sulla Terra, essendo la componente principale di ogni essere vivente e ha sempre svolto un ruolo fondamentale per lo sviluppo della civiltà umana. Nonostante l'acqua copra circa il 71% della superficie terrestre, più del 96% del totale è acqua salata. Il restante si trova perlopiù nei ghiacciai (circa il 68%), il 30% si trova sottoterra e la percentuale più piccola è nei laghi, nei fiumi e nell'atmosfera (Fig. 1.1) [1]. Per questo motivo, la gestione delle risorse idriche è cruciale per lo sviluppo della società nel rispetto dell'ambiente e della vita che lo abita. Più del 40% della popolazione mondiale è in condizioni di scarsità d'acqua, con più di 700 milioni di persone senza accesso ad acqua potabile. Circa il 50% delle acque reflue prodotte dall'uomo sono riversate direttamente in oceani o fiumi senza trattamenti, causando gravi danni all'ecosistema. Inoltre, le conseguenze del cambiamento climatico osservato negli ultimi decenni hanno aumentato l'urgenza del problema, viste le previsioni di crescenti disastri naturali come inondazioni. Questi scenari estremi causano a persone e strutture gravi danni difficili da riparare e possono incidere pesantemente sulla disponibilità e la qualità dell'acqua [7].

Per creare un'infrastruttura idrica sostenibile, si dovrebbe porre particolare attenzione alla conservazione e alla protezione dell'acqua. Esistono diverse tecniche per incrementare la disponibilità d'acqua, come la desalinizzazione, ma è importante migliorare anche le infrastrutture di raccolta, di conservazione e di distribuzione, nonché investire in un uso corretto delle risorse per evitare inquinamento e sprechi.

La costante ricerca di metodi innovativi e promettenti per la gestione dell'acqua ha recentemente portato ingegneri e ricercatori a sviluppare modelli di intelligenza artificiale (IA). L'intelligenza artificiale è stata applicata in vari modi per migliorare il monitoraggio, la previsione e l'utilizzo delle risorse idriche. I modelli di IA vengono addestrati su dati storici, seguendo approcci prevalentemente data-driven, che hanno la capacità di rilevare le relazioni non lineari tra input e

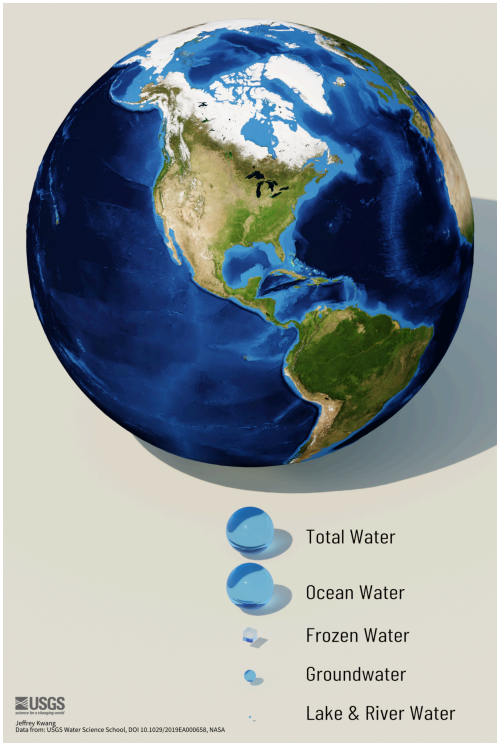


Figura 1.1: Distribuzione dell'acqua nel mondo [1].

output e dimostrano buoni risultati ed efficienza nelle previsioni [8].

Esistono numerosi esempi su come si può applicare l'IA nella gestione dell'acqua e dell'energia. Come nel caso di questa tesi, si può usare l'IA per analizzare dati storici in un sistema di distribuzione d'acqua o di un acquedotto, fornire acqua in modo ottimale e prevedere la domanda futura, riducendo i consumi di elettricità e i possibili sprechi d'acqua che si verificano durante la distribuzione.

Il settore agricolo pone una crescente domanda di acque da irrigazione, che si prevede raddoppierà entro il 2050. L'IA viene usata per ridurre questa crescita e per sviluppare sistemi di irrigazione intelligenti [9].

Un altro ambito d'interesse è il monitoraggio della qualità dell'acqua, particolarmente utile in Paesi con gravi problemi di inquinamento delle acque fluviali come in Africa, Asia e America Latina [9].

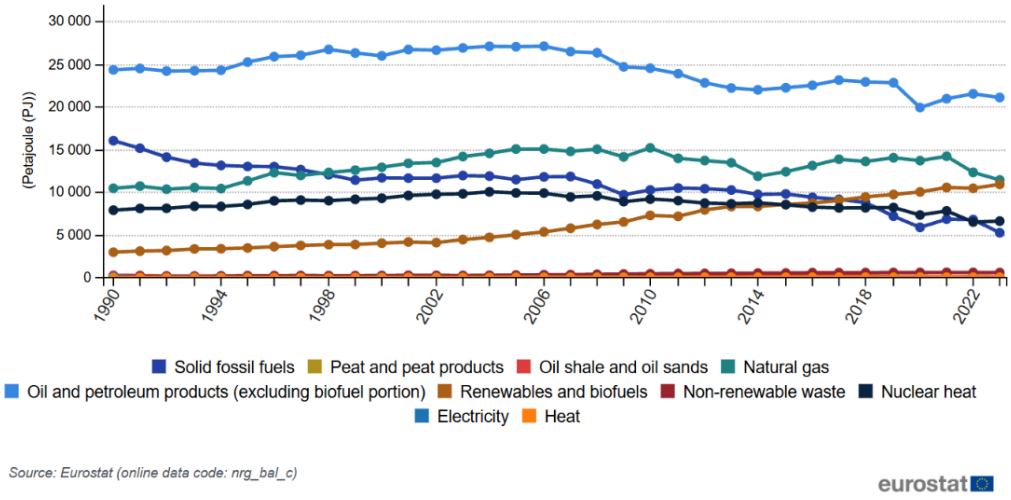


Figura 1.2: Consumo energetico nell'UE [2].

Come riportato in Figura 1.2, nel settore energetico si è osservato un incremento di utilizzo di energia da fonti rinnovabili, superando i combustibili fossili solidi nel 2018 e nel 2019, tuttavia il petrolio continua ad essere la fonte di energia principale nell'Unione Europea (37.6% nel 2023)[2]. L'intelligenza artificiale ha un ruolo importante nella costruzione di modelli di previsione di consumi energetici, essenziali per mitigare le conseguenze del riscaldamento globale e per avere prospettive aggiuntive durante l'analisi delle diverse situazioni climatiche nel mondo.

1.1 Motivazioni e Obiettivi

La scelta si svolgere questo lavoro di tesi è stata spinta dalla possibilità di dare un contributo concreto al miglioramento delle infrastrutture del territorio. Visto il contesto attuale è evidente che la salvaguardia dei beni primari come l'acqua è una priorità urgente e per vedere progressi a livello globale è importante impegnarsi anche a livello locale. I punti focali di questa tesi sono proprio la riduzione degli sprechi d'acqua e la minimizzazione dei consumi energetici prodotti durante la distribuzione. A questo proposito si sono impiegate tecniche innovative di intelligenza artificiale a basso costo computazionale come supporto decisionale al personale dell'acquedotto.
La presente tesi si inserisce infatti in un progetto in collaborazione con VIVA Servizi S.p.A. ed il Consorzio Gorgovivo. Lo scopo primario è quello di introdurre metodi di machine learning nei sistemi di gestione delle risorse idriche della sorgente per aumentarne l'efficienza e ridurne l'impatto energetico. Questo lavoro adotta un approccio data-driven, in cui le decisioni sono supportate da un'analisi accurata e oggettiva dei dati, anziché basarsi prevalentemente sull'esperienza o su valutazioni soggettive. Il vantaggio di quest'approccio è quello di poter costruire un modello che tiene conto del passato del sistema e che sia in grado di reagire dinamicamente a eventuali cambiamenti che si possono verificare nel tempo.

La sfida principale è comprendere correttamente tutti i dati coinvolti e manipolarli in modo da rispettare i requisiti richiesti, ossia organizzare l'attivazione dei pozzi in modo da ridurre il consumo energetico al minimo facendo sì che la richiesta d'acqua dei Comuni serviti da Gorgovivo sia sempre e comunque soddisfatta. Di fatto si tratta di un problema di ottimizzazione in cui l'obiettivo è minimizzare il costo energetico tenendo conto dei vincoli fisici del sistema.

Le tecniche di machine learning si inseriscono all'interno del problema nell'ambito di "Time Series Forecasting", ovvero si farà uso dello storico di dati per fare previsioni nel futuro che saranno fondamentali per l'ottimizzazione del sistema. Al termine di questa tesi si avrà un modello operativo che sarà la base su cui verrà implementato il sistema finale.

Il lavoro è stato quindi svolto in tre fasi:

1. **Raccolta dei dati**, ovvero la selezione dei dati storici corretti, che sono stati prodotti dalle misurazioni dei sensori posti nei pozzi della sorgente. Questi dati sono archiviati nei database dell'acquedotto e sono stati estratti per lo scopo di questo lavoro. Di questa mole di dati è necessario prenderne solo una parte per generare un unico dataset che poi verrà utilizzato per il resto delle operazioni.

2. **Analisi dei dati**, che consiste nella produzione di grafici e rapporti per

descrivere la natura dei dati e per comprendere il funzionamento dell'acquedotto. A partire da queste analisi si sono potute formulare delle strategie con cui utilizzare al meglio le informazioni ottenute in vista della fase finale del progetto.

3. **Ottimizzazione del sistema**, realizzata alla luce delle analisi svolte precedentemente. Si presenta come dei test svolti usando dati storici e implementando modelli di intelligenza artificiale per la previsione della richiesta idrica futura, da cui si determina l'allocazione di energia ottima per la sua soddisfazione. I risultati vengono esposti con dei grafici e con un benchmark per la comparazione di alcuni modelli di forecasting.

1.2 Struttura Tesi

Questa tesi è strutturata in modo tale da descrivere gradualmente gli elementi e i passaggi che compongono il progetto.

Il prossimo capitolo riguarderà il caso di studio Gorgovivo e conterrà tutte le informazioni necessarie per comprendere le caratteristiche del sito pertinenti a questo progetto, oltre che una panoramica sulla storia del complesso.

A seguire vi sarà un approfondimento su Time Series Forecasting e sull'evoluzione dello stato dell'arte del machine learning, oltre che dei riferimenti a dei lavori già esistenti.

Successivamente verranno descritte le tecnologie e i materiali impiegati nello sviluppo del progetto e infine seguirà una descrizione dettagliata della loro implementazione nel progetto.

In chiusura saranno discussi i risultati di questo lavoro e verranno poste delle considerazioni sulle prospettive future del progetto.

Capitolo 2

La sorgente Gorgovivo

2.1 Storia

Il complesso sorgentizio di Gorgovivo e le reti di adduzione sono stati istituiti e vengono amministrati dal Consorzio Gorgovivo, fondato nel 1963. Il Consorzio è attualmente composto da venti Comuni, di cui diciotto nella provincia di Ancona e due in provincia di Macerata. La gestione del servizio idrico è separata dal Consorzio e affidata ad un'apposita società, la VIVA Servizi S.p.A. [10].

Tra il 1970 e il 1979 furono svolte le opere di captazione, in cui vennero scavati 1.600 metri di gallerie e 13 camere con pozzi di roccia. Nel 1986 fu terminata la condotta adduttrice che parte da Serra San Quirico diramandosi nella Vallesina. Dalla condotta principale partono gli allacci alle reti idriche dei Comuni, a cui sono stati associati un totale di dodici serbatoi aggiuntivi che fungono da riserve d'acqua per i Comuni serviti. L'adduttrice principale di Gorgovivo serve 36 Comuni dell'AATO 2 (Assemblea Ambito Territoriale Ottimale n.2) per una popolazione di circa 345.000 abitanti, pari a circa l'86% della popolazione totale dell'AATO 2.

2.2 Descrizione sito

La sorgente si trova a valle della Gola della Rossa, nei pressi di Serra San Quirico in provincia di Ancona, Marche, Italia. Si sviluppa lungo la sponda destra del fiume Esino per un tratto di circa 1000 metri ed è costituita da numerose immissioni dalla roccia direttamente in alveo.

Il meccanismo della sorgente è il trabocco, posto contro il battente dei terreni alluvionali orientali, della falda immagazzinata nel calcare fratturato, alimentata per la massima parte dal bacino montano che si estende a partire dalla Gola della

Rossa oltre il massiccio del monte San Vicino, con una superficie complessiva di circa 200 chilometri quadri. Il complesso è costituito da due opere di presa:

- la principale, denominata "Gorgovivo", fu realizzata nei primi anni settanta e consiste di 12 pozzi tra loro interconnessi con una galleria di accesso di tipo ramificato; la galleria, che presenta l'imbocco in corrispondenza del "ponte canale" del ENEL, si sviluppa per circa due chilometri all'interno del massiccio calcareo del colle Sassi Rossi;
- la secondaria, denominata "Gorgovivo Bis", fu realizzata a metà degli anni novanta e consiste in una galleria "drenante" posta subito a valle della traversa Ex Montecatini; la galleria si sviluppa per circa 160 metri all'interno della coltre alluvionale delle ghiaie cementate che sovrastano la formazione calcarea.

I pozzi sono caratterizzati da dimensioni e capacità diverse, sono dotati di 2 pompe ciascuno (eccetto il pozzo 13 che ne ha 3) e sono posti ad altitudini differenti tra loro. Il flusso naturale della sorgente varia da un minimo di 1800 litri al secondo ad un massimo di 4000 litri al secondo [11].

L'acqua raccolta dai pozzi viene indirizzata verso il serbatoio principale, tramite un canale di circa 1200 metri di lunghezza, nella zona industriale di Serra San Quirico a 156 metri di altitudine sul livello del mare, un'altitudine tale da consentire di alimentare a gravità la maggior parte dei territori serviti. Il serbatoio funge da punto di snodo tra la sorgente e la rete di distribuzione dell'acqua che ha inizio dopo di esso. Infatti, la dimensione del serbatoio stesso è piuttosto contenuta, con un'altezza di 6 metri, rendendolo più adatto a scopi di misurazione dei livelli d'acqua anziché di conservazione a lungo termine. La quantità d'acqua al suo interno dipende solamente dalla portata in entrata e dalla portata in uscita.

Capitolo 3

Previsione di Serie Temporali

La previsione di serie temporali (time series forecasting) è un'operazione di estrema importanza in numerose applicazioni reali, come la previsione di mercati finanziari, di domanda di energia e monitoraggio climatico. Con l'aumentare della mole di dati collezionata in ogni ambito è aumentata la necessità di effettuare previsioni altamente accurate ed efficienti, che sono diventate essenziali come strumenti decisionali.

Questo capitolo fornisce una visione d'insieme sulle tipologie di time series e sull'evoluzione dei metodi più utilizzati nel forecasting, discutendo approcci moderni come il machine learning e le reti neurali. Dopo queste osservazioni verranno motivate le scelte fatte durante lo sviluppo del progetto per quanto riguarda le tecniche di analisi delle time series e i modelli utilizzati per il forecasting.

3.1 Serie Temporali

Le serie temporali (time series) sono sequenze di dati indicizzate dal tempo e ordinate cronologicamente. Si classificano in due tipologie:

- **Univariate**: contengono un singolo attributo misurato nel tempo
- **Multivariate**: composte da più attributi misurati negli stessi istanti

Per via del maggior numero di informazioni nelle serie multivariate, l'analisi e il forecasting tendono ad essere più complessi.

Le time series dispongono di diverse proprietà, alcune di queste si osservano tramite decomposizione della serie (Fig. 3.1).

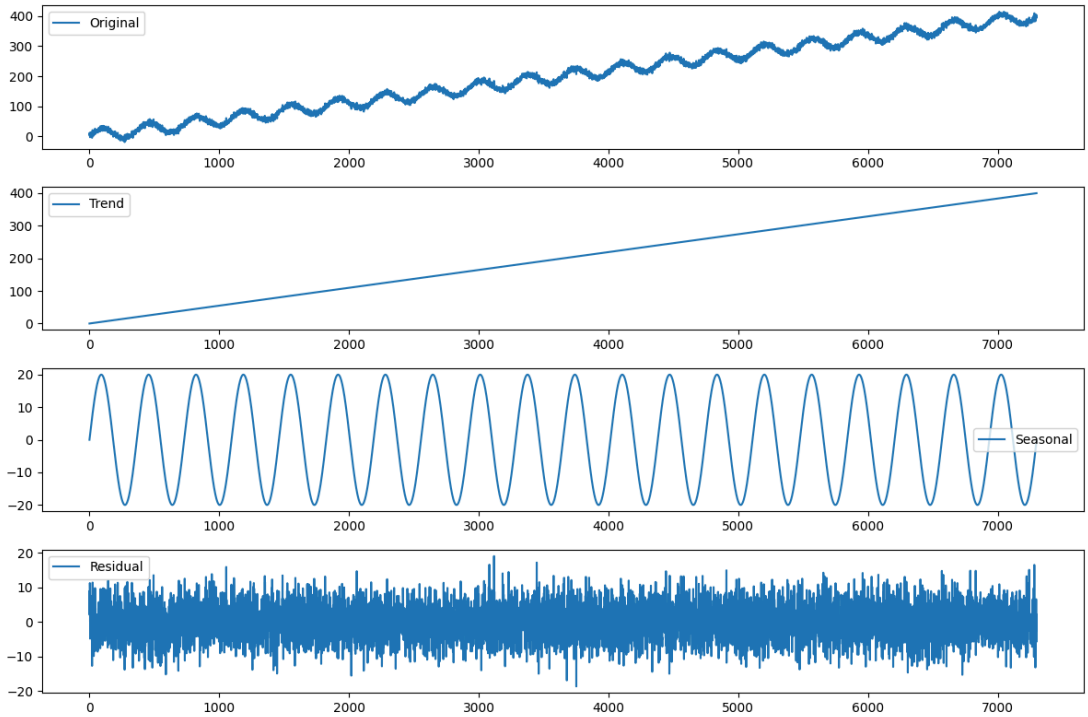


Figura 3.1: Decomposizione di serie di dati numerici sintetici [3].

- **Tendenza** (Trend): è il movimento a lungo termine della serie e può crescere o decrescere nel tempo. Per sequenze molto lunghe è possibile che il movimento possa alternarsi tra crescita e decrescita o possa anche appiattirsi.
- **Stagionalità** (Seasonality): indica comportamenti simili della serie durante certi archi temporali. Ad esempio, la temperatura dell'aria misurata nella stessa regione avrà un'alta stagionalità annuale, oppure la luce solare percepita da un pannello fotovoltaico avrà stagionalità giornaliera.
- **Residui** (Residuals): sono ciò che rimane della time series una volta rimossi i componenti di tendenza e stagionalità. Non c'è correlazione tra i residui.
- **Stazionarietà**: una serie temporale si dice stazionaria se non dipende dal tempo di osservazione, ovvero se traslandola avanti o indietro nel tempo la sua forma rimane invariata.
- **Autocorrelazione**: se un punto della serie ha una buona correlazione coi precedenti significa che ogni misurazione dipende da quelle precedenti.

3.1.1 Pre-processamento

La fase di pre-processamento (preprocessing) di time series ha lo scopo di preparare la serie in esame a diventare l'input della Exploratory Data Analysis (EDA). Durante il preprocessing è anche possibile iniziare a comprendere i dati che si stanno trattando, visualizzandoli così come sono, senza modifiche [12] .
Di seguito vengono introdotti i passaggi e gli aspetti fondamentali da tenere a mente durante il preprocessing.

3.1.1.1 Ordine cronologico

Per definire una time series è fondamentale assicurarsi che i dati siano stati correttamente misurati in ordine cronologico. Per farlo è sufficiente usare una funzione di ordinamento secondo i timestamp (l'indice temporale della serie).

3.1.1.2 Gestione dei valori mancanti

Durante la raccolta dei dati è possibile che si verifichino degli errori o dei guasti nei sensori che comportano la perdita di alcuni dati nella serie. È possibile rilevare e correggere questi errori con diversi gradi di accuratezza. I dati mancanti si riconoscono quando un attributo di una serie ha dei valori *"Bad"* o *"NaN"* (Not a Number) oppure quando presenta dei valori nulli che normalmente non dovrebbero esistere. Gestire questi valori mancanti è cruciale per avere una rappresentazione dei dati più vicina possibile alla realtà, per ottenere analisi veritiere e previsioni utili. Esistono diverse opzioni:

- Usare '0' o outlier: se il contesto lo consente è possibile sostituire i NaN con '0' o valori anomali come '-999'.
- Usare il valore medio: una soluzione semplice è quella di calcolare la media della serie e di approssimare i valori mancanti con la media.
- Forward-Filling: consiste nell'inserire l'ultimo valore valido in quello mancante.
- Interpolazione: usare un algoritmo di interpolazione può dare risultati sufficienti a basso costo computazionale. Utilizza i valori adiacenti ai NaN per riempirli.
- Imputazione: si può addestrare un modello di intelligenza artificiale sulla time series per riconoscere i valori mancanti e riempirli basandosi sull'andamento della serie. Può fornire risultati migliori ma in genere è un processo più complesso.

3.1.1.3 Ricampionamento

Il ricampionamento (resampling) è un algoritmo che può fornire informazioni aggiuntive sulla time series a seconda delle necessità. L'output dell'algoritmo è la stessa time series indicizzata in modo diverso. Si può svolgere in due modi:

- Upsampling: aumenta la frequenza di campionamento (e.g. da giorni a ore)
- Downsampling: riduce la frequenza di campionamento (e.g. da minuti a ore)

Inoltre, il resampling può essere usato per accertarsi che i timestamp della serie siano equidistanti tra loro, ovvero che le misurazioni siano state registrate a intervalli regolari.

3.1.1.4 Stazionarietà

In alcuni casi può essere utile verificare la stazionarietà della serie temporale. Alcuni modelli, come ARIMA (Autoregressive Integrated Moving Average), assumono che la serie sia stazionaria e non danno risultati corretti altrimenti [13].

Si definisce una serie stazionaria se ha:

- Media costante e indipendente dal tempo
- Varianza costante e indipendente dal tempo
- Covarianza costante e indipendente dal tempo

Di conseguenza una serie con tendenza e stagionalità non è stazionaria perché indicano rispettivamente che la media e la varianza non sono costanti nel tempo.

Si può testare la stazionarietà in tre modi:

- Visivamente: controllando i grafici della serie
- Tramite calcoli: dividendo la serie e calcolando media e varianza per ogni sezione
- Statisticamente: con test d'ipotesi (e.g. test di Dickey-Fuller [14])

3.1.1.5 Normalizzazione

È un processo con il quale si trasforma la scala della time series in una rappresentazione che va da 0 a 1. Si rivela particolarmente utile, se non necessario, nelle serie multivariate con parametri in scale diverse che devono essere usati per addestrare modelli di intelligenza artificiale. È inoltre utile per visualizzare e confrontare time series di cui si vuole calcolare la correlazione. Gli algoritmi di normalizzazione richiedono la possibilità di calcolare o stimare i valori massimi e minimi della time series.

Un valore x si può normalizzare in questo modo:

$$y = (x - min)/(max - min)$$

Dove i valori max e min appartengono alla distribuzione di x che va normalizzata.

3.1.1.6 Standardizzazione

Quando si sta standardizzando una serie, si sta scalando la distribuzione dei valori in modo da avere media nulla e deviazione standard unitaria. La standardizzazione ha come requisito che la serie si avvicini a una distribuzione Gaussiana con media e deviazione standard ben definite. Con questo tipo di distribuzione, standardizzare i dati può dare risultati migliori nelle previsioni.

Un valore x si può standardizzare in questo modo:

$$y = (x - mean)/standard_deviation$$

Dove media e deviazione standard appartengono alla distribuzione di x da standardizzare.

3.1.2 Analisi Dati Esplorativa

L'analisi dati esplorativa (EDA) è la fase in cui si producono rapporti e grafici dei dati raccolti. Al termine di questa fase si deve avere abbastanza materiale da poter descrivere il sistema soltanto con i dati a disposizione nel modo più comprensivo possibile. I risultati ricercati durante l'EDA variano da caso a caso ed esistono diversi metodi e tecniche per ottenerli. In genere si vogliono rappresentare le time series con dei grafici che possano rappresentare al meglio le loro caratteristiche, come per esempio diagrammi a linea, a barre, a dispersione o mappe di calore (heatmap).

Oltre alla costruzione di grafici, si rivela di grande utilità eseguire calcoli e impiegare algoritmi per ottenere maggiori informazioni e confrontare serie diverse.

3.1.3 Correlazione

In statistica, la correlazione è una relazione tra due dati variabili. In genere ci si riferisce a relazioni di tipo lineare. All'atto pratico, la correlazione può essere utile per individuare legami tra time series che possono essere sfruttati per ottenere previsioni migliori. È possibile misurare la correlazione con diversi coefficienti, tra cui uno dei più comuni è il coefficiente di Pearson [15].

Il coefficiente di Pearson è un numero reale che può assumere valori compresi tra –1 e 1 che indica l'intensità e il tipo di relazione lineare tra due variabili aleatorie. Si calcola come il rapporto tra la covarianza delle due variabili aleatorie e il prodotto delle loro deviazioni standard.

- Se il coefficiente ha valore nullo, non c'è correlazione tra le variabili.
- Se ha valore pari a 1, c'è una relazione lineare totale tra le variabili, ovvero se una variabile cresce, l'altra cresce esattamente allo stesso modo.
- Se ha valore pari a –1, c'è una relazione inversamente proporzionale, ovvero se una variabile cresce, l'altra decresce in maniera esattamente opposta.

Nonostante l'utilità di questo coefficiente, presenta delle vulnerabilità. È in grado di rilevare correlazione tra due distribuzioni soltanto se non c'è ritardo tra due grandezze fisiche. In più, dover assumere relazioni soltanto di tipo lineare può rappresentare un limite in applicazioni reali con grandezze non lineari [16] .

3.1.3.1 Dynamic Time Warping

Dynamic Time Warping (DTW) è un algoritmo usato per confrontare time series per trovare similarità, che può essere utile per far fronte ai limiti della correlazione tradizionale. Lo fa spostando punti di dati nel tempo in modo da minimizzare la

differenza tra i due dataset. È capace di confrontare sia serie con stessa lunghezza che di diverse lunghezze, infatti è spesso utilizzato in ambiti dove si osservano differenze tempistiche evidenti, come il riconoscimento vocale, analisi dei movimenti e finanza.

L'algoritmo si svolge in quattro passi:

1. **Matrice Distanze**: il primo passo consiste nel costruire la matrice delle distanze, dove ogni elemento rappresenta la distanza (Euclideana) tra due punti corrispondenti nelle due sequenze.
2. **Matrice Costi**: successivamente viene creata la matrice dei costi accumulando le distanze minime dall'inizio della sequenza fino al punto corrente. Il costo accumulato rappresenta la distanza cumulativa del percorso ottimale fino a quel punto.
3. **Percorso Ottimale**: si trova il percorso di allineamento ottimale procedendo a ritroso dall'ultimo al primo elemento della matrice dei costi. Questo è il percorso che minimizza la distanza totale tra le due serie.
4. **Modifica Percorso**: il percorso ottimale viene allungato o compresso in modo da corrispondere all'asse temporale dell'altra sequenza.

DTW ha come vantaggi la possibilità di analizzare sequenze di dati di diverse lunghezze e con strutture complesse, non lineari e in ambienti rumorosi, dove altri metodi potrebbero fallire.

I limiti di DTW risiedono nella sua complessità computazionale (quadratica rispetto alla lunghezza delle sequenze), infatti lunghe sequenze possono richiedere una grande mole di calcoli. Per tamponare questo problema è stato definito il metodo "Sakoe-Chiba" [17]. Esso consiste nel definire la dimensione della finestra di computazione di DTW, che indica la più grande traslazione temporale consentita dalla diagonale in direzione della time series più lunga. Una finestra più piccola richiede minore costo computazionale ma dà risultati peggiori, la Figura 3.2 mostra la finestra in tre scenari diversi e con tre dimensioni diverse. Il metodo "Sakoe-Chiba" può anche ridurre over-warping irrealistici che si possono verificare in assenza di limitazioni [4].

Per utilizzare DTW è necessario normalizzare le time series in quanto non è sensibile a scale eterogenee nei dati.

3.1.3.2 Autocorrelazione

Con funzione di autocorrelazione (ACF) si intende una funzione che misura la correlazione dei punti della time series con i precedenti.

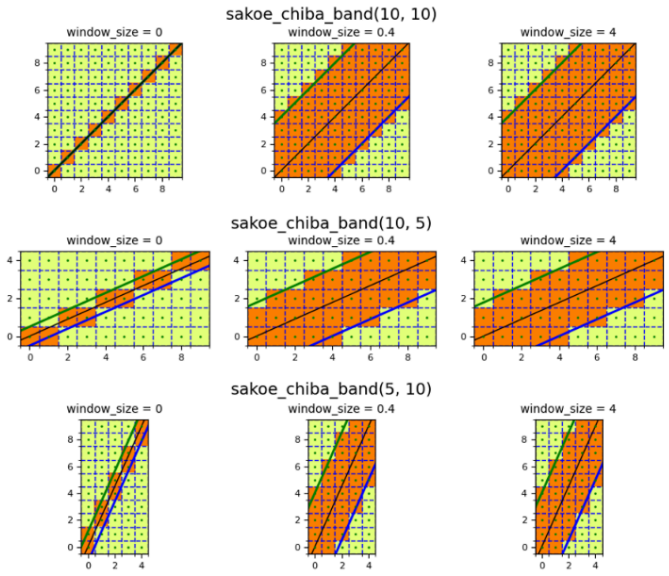


Figura 3.2: Finestra di Sakoe-Chiba [4].

Data una time series $x_t = \{x_0, ..., x_t\}$, si misura la correlazione tra un punto x_t e x_{t-k} con $k = 1, 2, ...$

Questo tipo di correlazione cattura la relazione lineare presente nei punti della serie stessa e può essere utile nella fase di tuning dei modelli di forecasting.

La funzione di autocorrelazione parziale (PACF) è una variante che misura la correlazione tra due punti nella serie senza considerare il contributo lineare dei punti intermedi. Ad esempio, per rimuovere l'influenza del valore intermedio per $k = 2$, si calcola la correlazione tra $x_t - \beta_0 * x_{t-1}$ e $x_{t-2} - \beta_1 * x_{t-1}$, dove β_0 e β_1 si ottengono regredendo linearmente x_t su $x_t - 1$ e x_{t-2} su x_{t-1}.

PACF è particolarmente utile per determinare l'ordine di autoregressione (AR) nei modelli ARIMA.

È importante inoltre notare che ACF e PACF rilevano soltanto relazioni lineari; quindi, se non dessero buoni risultati potrebbe significare che sono presenti soltanto relazioni non lineari all'interno della serie.

3.1.3.3 Filtro di Hampel

Quando una time series presenta diversi outlier (valori anomali) che non si vogliono includere nell'analisi è possibile rimuoverli, o attenuarli, sfruttando un filtro

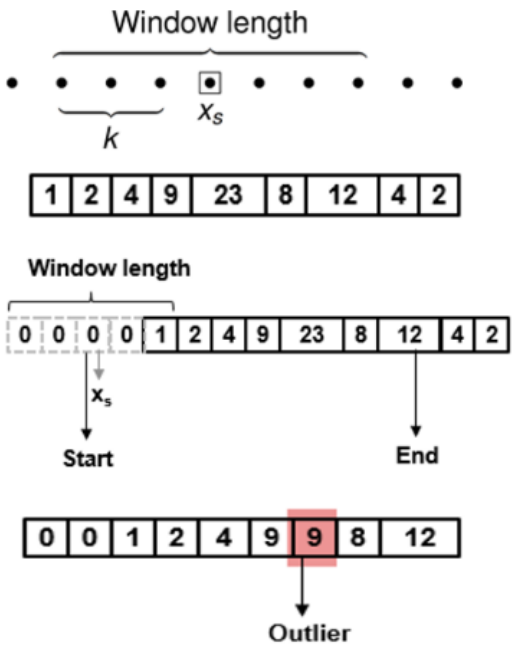


Figura 3.3: Filtro di Hampel [5].

come il filtro di Hampel.

Il filtro di Hampel rileva e rimuove gli outlier nel segnale in input applicando un algoritmo sulla serie.

Dato un campione x_s , l'algoritmo:

1. Centra una finestra di lunghezza dispari sul campione corrente
2. Calcola la mediana locale m_i e la deviazione standard σ_i sulla finestra.
3. Confronta il campione con $n_\sigma * \sigma_i$ dove n_σ è il valore di soglia.
   Se $|x_s - m_i| > n_\sigma * \sigma_i$, il campione è un outlier e il filtro lo sostituisce con la mediana.

Si consideri una sequenza di dati da filtrare con una finestra di lunghezza 5 e un valore di soglia pari a 2. La finestra scorre lungo la sequenza e per ogni valore si esegue l'algoritmo, fino a che la finestra non viene centrata sull'ultimo campione. A questo punto il filtro genera l'output.

3.2 Previsione

La previsione (forecasting) è il processo attraverso cui si cerca di stimare l'andamento futuro sulla base delle informazioni disponibili, riducendo il più possibile il margine di errore. Nel caso del time series forecasting, l'obiettivo è prevedere i campioni successivi alla fine della serie temporale osservata. In base al numero di variabili considerate, il forecasting può essere classificato come univariato o multivariato.
Nel forecasting univariato viene considerata una sola variabile per effettuare la previsione. Un esempio è la previsione dell'andamento della popolazione mondiale nel tempo.
Il forecasting di serie multivariate invece considera più variabili (tutte time series) in relazione tra loro. Si può trasformare l'esempio precedente in un caso di forecasting multivariato con l'aggiunta di altri parametri correlati alla popolazione come tasso di natalità, tasso di mortalità e migrazione internazionale [18].

3.2.1 Applicazioni

Il forecasting, in questo caso di time series, è impiegato in una vasta gamma di applicazioni reali in cui è necessario avere informazioni riguardo il futuro per effettuare pianificazioni a breve e a lungo termine. Di seguito alcuni esempi di queste applicazioni [19]:

- **Settore energetico**: con la trasformazione dell'industria energetica verso un utilizzo crescente di fonti rinnovabili e di gestione efficiente e sostenibile delle risorse, il corretto impiego del forecasting può rivelarsi critico. La produzione, la distribuzione ed il consumo di energia possono essere significativamente migliorati con sistemi di ottimizzazione che fanno uso di previsioni e di dati storici. Questa tesi si inserisce in questo contesto con l'intento di ridurre l'impatto energetico dovuto alla distribuzione delle risorse idriche della sorgente Gorgovivo [20].

- **Settore finanziario**: nella finanza, si usa frequentemente il forecasting per prevedere cicli di business e movimenti di mercato a lungo termine. Può assistere investitori nella decisione di strategie di investimento e può supportare le aziende nel pianificare i budget, gestire il rischio e monitorare le prestazioni aziendali [21].

- **Monitoraggio ambientale**: i forecast ecologici prevedono cambiamenti negli ecosistemi e nei loro componenti, conseguenza di condizioni meteorologiche estreme, inquinamento o cambiamenti negli habitat. Se svolte correttamente, queste previsioni possono essere usate dalle autorità locali per proteggere aree a rischio e notificare sistemi di allerta meteo. [22] Inol-

tre, strategie di questo tipo sono comunemente impiegate anche in contesti industriali, come agricoltura e trasporto marittimo.

- **Settore farmaceutico**: il forecasting viene applicato in diverse fasi nello sviluppo farmaceutico. Previsione di tossicità di un medicinale, farmacocinetica, farmacodinamica e altri parametri facilita i processi di progettazione e monitoraggio. Trova applicazioni anche lato marketing, nella previsione di vendite di farmaci specifici e di prospettive di mercato [23].

3.2.2 Previsione di Serie Temporali Multivariate

Il forecasting di serie temporali multivariate richiede delle accortezze aggiuntive rispetto al caso a singola variabile, infatti necessita della selezione delle variabili più adatte al caso di studio e dell'individuazione di eventuali dipendenze tra di loro. Inoltre, sono richieste accortezze in merito alla decisione dell'orizzonte di forecasting e alla verifica dell'interpretabilità dei risultati.

3.2.2.1 Selezione delle variabili

Le time series multivariate contengono, come suggerisce il nome, diverse variabili ordinate nel tempo. Durante il forecasting si vogliono sfruttare le relazioni che legano queste variabili per ottenere previsioni più efficaci. Tuttavia, la serie potrebbe contenere anche variabili che non mostrano dipendenze con le altre e che quindi non vanno incluse nella fase di forecasting. Le dipendenze tra variabili possono manifestarsi a breve o a lungo termine. Nelle dipendenze a breve termine il valore corrente di una variabile dipende dai valori più recenti della variabile stessa e delle altre; a lungo termine sono presenti fattori che determinano il comportamento globale di più variabili [24].

L'individuazione di dipendenze tra variabili è facilitata da un'estensiva analisi dei dati del dominio.

3.2.2.2 Orizzonte di previsione

L'orizzonte di previsione (o di forecasting) è l'arco di tempo nel futuro lungo il quale va fatta una previsione, che varia a seconda del problema specifico e va scelto in base al periodo utile della previsione. L'orizzonte di forecasting influenza l'accuratezza e l'affidabilità della previsione. In genere più si cerca di prevedere un futuro lontano e più si introduce errore nella previsione.

Esistono due tipi di orizzonte (horizon) [24]:

- Point forecasting: usato per fare una singola previsione di un valore futuro. Un limite di questo metodo è che assume l'esistenza di un processo che

genera i dati della time series in maniera immutabile. Spesso non è il caso di molte applicazioni reali.

- Multi-Horizon forecasting: prevede diversi punti nel tempo lungo un orizzonte predeterminato. All'espandersi dell'orizzonte spesso si associa una degradazione della qualità del forecast.

3.2.2.3 Interpretabilità

Si riferisce alla facilità di comprensione dei fattori da cui dipendono le previsioni prodotte da un modello. Una buona interpretabilità aiuta gli analisti a prendere decisioni e a raffinare l'accuratezza del modello di forecasting. Il corretto uso delle tecniche sopra descritte e l'aggiunta di visualizzazioni e sistemi di riconoscimento di pattern nei dati aiutano a migliorarne l'interpretabilità [24].

3.2.2.4 Metriche di valutazione

Esistono diverse metriche per valutare le prestazioni di un modello di forecasting su una time series:

- Mean Absolute Error (MAE) e Root Mean Squared Error (RMSE): misurano la media assoluta e le differenze quadrate tra i valori reali e quelli previsti per ogni variabile.
- Mean Absolute Percentage Error (MAPE): misura la differenza media in percentuale tra i valori effettivi e quelli previsti per ogni variabile.
- Mean Square Error (MSE) e R-squared (R2): MSE è la media degli errori al quadrato tra valori reali e previsti. R2 misura la proporzione della varianza totale in tutte le variabili del modello.

Il confronto tra dati reali e previsti si può fare dividendo il dataset in train-set e test-set. Il modello viene addestrato sul train-set e il test-set viene usato in fase di forecast come ground-truth (verità assoluta), su cui viene verificata l'efficacia delle previsioni. Dopo aver testato il modello sul test-set e valutata la sua accuratezza, si possono fare forecast effettivi sul futuro incognito [24].

3.2.3 Metodi Statistici

Nel time series forecasting esistono molti metodi per fare previsioni, tra cui metodi statistici, machine learning e deep learning. I metodi statistici usano modelli matematici, mentre machine learning e deep learning usano tecniche di apprendimento automatico.

I metodi statistici usano rappresentazioni matematiche delle time series per prevederne il comportamento futuro. Lo stesso modello statistico produrrà sempre lo stesso output per lo stesso input. Esistono molti modelli statistici, tra cui Auto-Regressive (AR), Moving Average (MA), Auto-Regressive Moving Average (ARMA) e Auto-Regressive Integrated Moving Average (ARIMA).

3.2.3.1 ARMA

Il modello Auto-Regressivo a Media Mobile è la combinazione dei più semplici AR e MA e può rappresentare time series stazionarie.

Il componente AR usa la dipendenza tra l'osservazione attuale e da p osservazioni precedenti per prevedere il prossimo valore. p è detto ordine di auto-regressione.

Il componente MA usa la dipendenza tra l'osservazione corrente e l'errore residuo da un modello MA applicato a q osservazioni precedenti. q è l'ordine del componente a Media Mobile.

È un modello relativamente semplice da implementare e può anche dare buoni risultati per bassi valori di p e q, però può funzionare soltanto con serie stazionarie.

3.2.3.2 ARIMA

Il modello ARIMA si differenzia dal modello ARMA per la presenza di una componente di integrazione, definita dal grado di differenziazione d. Questa componente I indica che i valori vengono rimpiazzati dalla differenza tra ogni valore ed il suo precedente. Questa operazione ha lo scopo di stabilizzare la media e rendere la tendenza della serie stazionaria, ma non affligge la non stazionarietà della varianza. Ciò vuol dire che se la serie originale ha varianza costante ma media variabile, la si può approssimare come serie stazionaria, rispettando il requisito del modello. Esistono numerose estensioni e varianti di questo modello [13].

Alcuni tipi classici di modelli $ARIMA(p, d, q)$ sono:

- $ARIMA(1, 0, 0)$ = modello auto-regressivo di primo ordine: se la serie è stazionaria e autocorrelata, la si potrebbe prevedere come multiplo dei valori precedenti più una costante.
- $ARIMA(0, 1, 0)$ = random walk: se la serie non è stazionaria, il modo più semplice per rappresentarla è usando una lista di numeri, dove ognuno è una variazione del precedente.
- $ARIMA(1, 1, 0)$ = modello auto-regressivo di primo ordine differenziato: se gli errori di un modello random walk sono autocorrelati, si può tentare

di risolvere il problema regredendo il valore corrente su sé stesso traslato di uno step.

- $ARIMA(0, 1, 1)$ = exponential smoothing: usato per time series senza stagionalità o tendenza. Usa un parametro di attenuazione (con valori tra 0 e 1) per controllare l'influenza di ogni osservazione storica.

3.2.4 Metodi di Machine Learning

Il termine machine learning è usato per definire una branca all'interno del mondo dell'intelligenza artificiale che si pone l'obiettivo di far apprendere alle macchine delle attività reali. Un algoritmo di machine learning si distingue da un algoritmo tradizionale dal fatto che il computer può imparare e migliorare la performance grazie all'esperienza. Quindi, il machine learning ha l'obiettivo di creare modelli che permettono di costruire algoritmi di apprendimento in vari modi a seconda dello scopo dell'anailisi. Durante lo sviluppo del progetto di tesi si sono favorite tecniche di machine learning per la relativa semplicità di utilizzo e basso costo computazionale.

Esistono diversi tipi di apprendimento:

Apprendimento supervisionato È un algoritmo che prende X dati di input e Y dati di output e che apprende la funzione che dall'input genera l'output. Si vuole approssimare la funzione reale in modo da poter prevedere l'output dato un input. L'apprendimento si interrompe quando si ottiene una prestazione accettabile.

Alcuni problemi di apprendimento sono:

- Classificazione: la macchina deve automaticamente riconoscere e categorizzare oggetti presi da un dataset.
- Regressione: si vuole prevedere un valore in base ai dati a disposizione. È il tipo di problema affrontato in questa tesi.

Apprendimento non supervisionato Quando si ha a disposizione solo la variabile di input X e nessuna variabile di output. Si cercano relazioni tra i dati analizzati senza avere la risposta corretta come riferimento, da cui il nome "non supervisionato".

I problemi non superivisionati possono essere:

- Raggruppamento: l'algoritmo cerca di individuare relazioni tra i dati per raggruppare quelli con caratteristiche simili.

- Associazione: si cerca di scoprire regole e strutture causali in un insieme di oggetti in un database relazionale. Strettamente legato al Data Mining.

Apprendimento con rinforzo Una tecnica che rappresenta l'apprendimento per tentativi ed errori. Usa un sistema di valutazione, in cui si fornisce una ricompensa in caso di decisioni corrette e una penalità in caso contrario. Si vuole massimizzare la ricompensa ottenuta senza dover annunciare la strada da prendere.

3.2.4.1 Regressione

In questo progetto si è usato il machine learning per risolvere un problema di forecasting, che rientra nella categoria di apprendimento supervisionato detta regressione. Esistono molti modelli sia statistici che di machine learning ben conosciuti e testati per la soluzione di problemi di regressione, tra cui:

- **Regressione Lineare**: un metodo statistico fondamentale usato anche per machine learning nello studio di relazioni tra input e output numerici. Rappresenta un'equazione lineare che combina x input in modo che l'output previsto sia y. Si assegna all'input un coefficiente β come fattore di scala e si aggiunge un altro coefficiente per un grado di libertà in più.

  $$y = \beta_0 + \beta_1 * x$$

  In iperpiani a più dimensioni si usano più variabili di input. Durante l'apprendimento si stimano i valori dei coefficienti usando i dati a disposizione. Il modo più semplice è usare la statistica, ovvero calcolando le proprietà dei dati come media, varianza, correlazione e covarianza.

  Per fare una previsione essenzialmente si deve risolvere un'equazione per un insieme specifico di input, dopo aver stimato i coefficienti con una tecnica di apprendimento.

  La regressione lineare è un modello molto robusto e veloce, tuttavia essendo lineare non può catturare relazioni non lineari nei dati, che può essere un limite in situazioni reali.

- **Regressione Polinomiale**: si distingue dalla regressione lineare per la presenza di un polinomio di grado d, ottenendo questa forma.

  $$y = \beta_0 + \beta_1 * x + \beta_2 * x^2 + ... + \beta_d * x^d$$

  Può modellare curve mantenendo la struttura lineare dei parametri ma all'aumentare del grado d diventa estremamente sensibile all'overfitting (si ha quando l'algoritmo tende a memorizzare i dati dell'addestramento senza imparare i pattern).

- **Alberi Decisionali per Regressione** (Decision Tree Regression): suddivide ricorsivamente i dati in iperpiani ortogonali paralleli agli assi coordinati e assegna a ciascuna foglia dell'albero il valore medio dei campioni ottenuti. È facilmente interpretabili graficamente, ma produce previsioni a gradino e tende ad avere alta varianza.

- **Random Forest**: usa un insieme di alberi decisionali indipendenti addestrati in parallelo con Bootstrapping (campionamento casuale con reinserimento) e Feature Bagging (scelta casuale di un sottoinsieme di feature a ogni split). Il risultato finale è la media di tutti gli alberi. È molto robusto ed efficace su dataset complessi. Alto consumo di memoria.

- **XGBoost** (eXtreme Gradient Boosting): algoritmo sequenziale dove ad ogni step si addestra un nuovo albero per correggere gli errori dello step precedente. Richiede un tuning accurato degli iperparametri per risultati migliori.

3.2.5 Deep Learning

Il deep learning è un ramo del machine learning che fa affidamento su reti neurali multistrato per svolgere il processo di apprendimento. Il vantaggio del deep learning è che può funzionare particolarmente bene con dati multidimensionali estremamente complessi, riuscendo a estrarre relazioni e pattern anche non lineari e a lungo termine. Esistono molti modelli per fare time series forecasting ed è una tecnologia in continua evoluzione [25].

- **AutoEncoder** (AE): un modello di apprendimento non supervisionato usato perlopiù in operazioni di rilevamento. Consiste in un encoder e un decoder. L'encoder mappa l'input in uno spazio a poche dimensioni, mentre il decoder cerca di ricostruire i dati originali. È un'architettura semplice che dà buoni risultati con diversi tipi di dato. Il rischio di overfitting si alza all'aumentare della complessità dei dati o con pochi campioni di base. Per superare queste limitazioni sono state create delle varianti come il Variatonal AutoEncoder (VAE), che introduce modelli probabilistici.

- **Convolutional Neural Network** (CNN): è una classe di reti neurali incentrate su operazioni di convoluzione, inizialmente impiegate nella Computer Vision. La convoluzione serve a rilevare cambiamenti nei dati storici nel tempo, usati per fare previsioni. Tuttavia, si rivela essere poco sensibili a cambiamenti dinamici nelle time series, risultando in alcuni casi in performance scarse. Questo modello funziona bene in ambito finanziario, essendo semplici da addestrare e resistenti a dati rumorosi.

- **Recurrent Neural Network** (RNN): reti costituite da anelli collegati ricorsivamente in direzione dello sviluppo della sequenza. Dimostrano grandi

capacità di apprendimento, riconoscendo fattori sequenziali, ma data la presenza di legami ricorsivi diventa molto vulnerabile a problemi di gradient exploding (gli errori commessi ad ogni step si accumulano in modo incontrollabile). Per risolvere molti problemi delle RNN, è stata introdotta la variante Long-Short-Term-Memory (LSTM). Ogni cella della rete contiene un *input gate*, un *forget gate* e un *output gate*. Il *forget gate* decide quante informazioni mantenere dallo step precedente, moltiplicando lo stato precedente per un valore tra 0 e 1. L' *input gate* moltiplica lo stato corrente per un valore tra 0 e 1 per stabilire quante informazioni aggiungere alla memoria. Infine l'*output gate* decide quale parte dello stato interno deve rimanere visibile come output e quale deve diventare un nuovo stato nascosto. È un modello molto utilizzato in grado di dare buoni risultati, ed esistono molte varianti ed estensioni [26].

- **GAN** (Generative Adversarial Network): costituiti da un generatore, che impara a produrre campioni simili a quelli reali e da un discriminatore, che impara a distinguere dati reali da quelli sintetici. Usati per costruire dati fittizi realistici e di qualità.

- **Transformer**: un modello proposto nel 2017 basato su un meccanismo di auto-attenzione, abbandonando quindi le architetture a CNN e RNN. È estremamente efficiente nel gestire calcolo in parallelo e dati di dimensioni enormi, ma è limitato da un alto costo computazionale. Usato nell'analisi del linguaggio naturale e nel time series forecasting.

Tutti questi modelli dispongono di caratteristiche diverse, vantaggi e svantaggi e dovrebbero essere scelti in base al campo applicativo, costo computazionale, dimensione del dataset, scalabilità, benchmark e tipologia di dati.

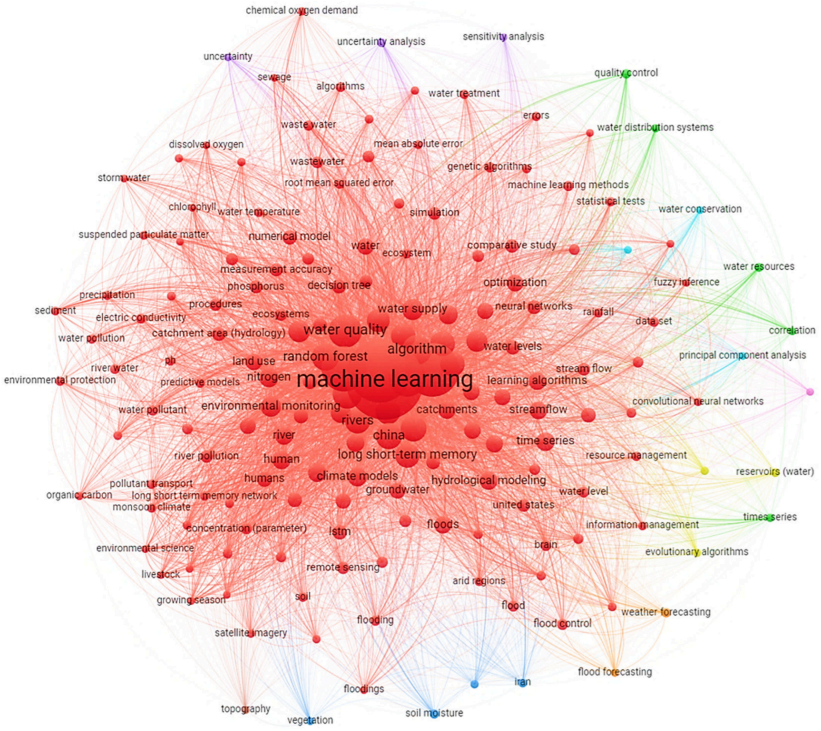


Figura 3.4: Parole chiave su applicazioni di ML per risorse idriche [6].

3.3 Lavori Correlati

I numerosi progetti di ricerca che sono stati svolti nell'ambito machine learning per la gestione delle risorse idriche danno valore a queste tecnologie innovative, che ad oggi sono diventate cruciali in tantissimi settori. La Figura 3.4 mostra le occorrenze delle parole chiave nella letteratura su applicazioni di ML per risorse idriche.

Nel 2023, (Ghobadi e Kang [6]) è stata compilata una ricerca sistematica sulle applicazioni del machine learning per la gestione delle risorse idriche, usando una vasta gamma di database accademici in modo da comprendere più aree di studio possibili. Lo studio è stato categorizzato secondo i tre principiali gruppi di algoritmi di ML (supervisionati, non supervisionati e rinforzati). Il forecasting si trova in vari casi d'uso come: richiesta d'acqua, qualità dell'acqua, produzione di energia idroelettrica, monitoraggio livelli d'acqua.

Zanfei et. [27] al hanno messo a confronto alcune reti neurali convoluzionali con architetture a grafo (GCRNN), LSTM e SARIMA nella previsione della richiesta d'acqua. Dai risultati si osserva che, in base alle metriche di valutazione, le GCRNN hanno avuto prestazioni leggermente migliori di LSTM e sempre migliori del modello SARIMA.

Il clustering viene usato in situazioni simili, perlopiù per la classificazione di acque sotterranee, pluviali e fluviali, monitoraggio dei sistemi di distribuzione e di pattern nei consumi d'acqua.

L'apprendimento rinforzato è un sottoinsieme del machine learning che ha recentemente guadagnato attenzione e può essere usato per simulare le decisioni umane nella gestione di sistemi naturali e socioecologici complessi. Essendo una tecnologia relativamente recente, ancora non ha potuto crescere in popolarità nel campo dell'idrologia, ma ci si aspetta che verrà impiegata in un numero crescente di applicazioni reali in futuro [6].

Nel 2019, Rozos [28] ha svolti analisi ed esperimenti sull'acquedotto di Atene, Grecia usando *feedforward neural network* (FFN), con lo scopo di sfruttare questo strumento come un supporto decisionale che possa descrivere il funzionamento dell'intero sistema di distribuzione d'acqua. La rete è stata addestrata secondo quattro politiche diverse, con dati sintetici. I risultati dei test dimostrano che la rete fosse in grado di gestire il sistema, riproducendo con buoni livelli di accuratezza i dati reali, come i livelli dei serbatoi e le estrazioni d'acqua, con una probabilità massima di fallimento annuale stimata del 2%. L'ottimizzazione del sistema con questo metodo facilita inoltre l'integrazione di altri modelli per via della sua semplicità e velocità di esecuzione.

Capitolo 4

Materiali e Metodi

Lo svolgimento del progetto ha richiesto l'integrazione di diversi strumenti per creare un ambiente di sviluppo con cui è possibile elaborare i materiali di base secondo gli scopi che sono stati posti. È stato anche importante valutare quali piattaforme utilizzare per poter esporre in modo chiaro e intuitivo i risultati ottenuti dopo le analisi.

In questo capitolo vengono presentate tutte le tecnologie alla base di questo progetto, i materiali che sono stati raccolti e il metodo che si è seguito per il suo sviluppo.

4.1 Ambiente di sviluppo

4.1.1 Python

Python è un linguaggio di programmazione ad alto livello largamente utilizzato in ambito Data Science e Machine Learning. Le caratteristiche di Python sono quelle di essere un linguaggio interpretato, general-purpose e multi-paradigma con una sintassi molto semplice rispetto ad altri linguaggi di programmazione. La vasta gamma di librerie disponibili rendono Python una buona scelta per lavorare con grandi dataset e per implementare modelli di machine learning.

Alcune delle librerie utilizzate sono:

- Pandas: permette di elaborare dati conservati in spreadsheet (e.g. file ".xlsx" o ".csv") o database inserendoli in tabelle chiamate "DataFrame". I DataFrame sono matrici dove ciascuna colonna è identificata da una label e ogni riga da un index. La libreria contiene diverse funzioni per elaborare e visualizzare i dati contenuti nei DataFrame.
- NumPy: aggiunge supporto per operazioni di algebra lineare su vettori e matrici, necessaria come dipendenza per numerose altre librerie.

- Sci-kit learn: libreria di machine learning con un gran numero di funzioni di supporto per le fasi di preprocessing dei dati e di fitting e valutazione dei modelli di machine learning. Contiene anche modelli di machine learning come Random Forest.
- Matplotlib: vasta libreria di visualizzazione dati che consente di creare grafici di vario tipo e di salvarli in immagini lavorando con DataFrame.
- Seaborn: basata su Matplotlib, espande funzionalità su grafici statistici.
- os: modulo di Python usato per svolgere operazioni di gestione del file system del sistema operativo, come aprire o salvare file.
- re: modulo di Python che fornisce operazioni con regular expressions (stringhe con caratteri speciali), per confronti e ricerche di stringhe particolari.
- glob: modulo di Python usto per trovare pathname di cartelle e file.
- datetime: gestisce formati temporali (data e ora).

4.1.2 Visual Studio Code

Durante la fase di sviluppo il codice è stato scritto e testato nell'IDE Visual Studio Code, utilizzando gli strumenti di supporto per Python come il syntax highlighting e il debugger.

4.1.3 Google Colab

Per la visualizzazione dei dati e del codice si è utilizzato Google Colab, un servizio cloud gratuito che permette di scrivere codice Python in notebook Jupyter.

I notebook sono dei documenti costituiti da celle che possono contenere testo o codice. Le celle di codice possono essere eseguite grazie all'infrastruttura cloud di Colab, che predispone lo sviluppo in Python, sfruttando le risorse allocate in base al proprio piano di abbonamento ed è possibile salvare l'output delle celle per poterlo visualizzare anche dopo aver interrotto la connessione con il runtime. Le celle testuali sono usate per aggiungere descrizioni del codice e informazioni aggiuntive; questo rende i notebook un buono strumento di condivisione delle informazioni.

Colab è stato utile per presentare il lavoro svolto al personale di Gorgovivo e VIVA Servizi S.p.A. e per condividere il codice senza problemi riguardanti gli ambienti virtuali e le dipendenze tra librerie.

4.1.4 Microsoft Excel

I dati in input sono organizzati in fogli di calcolo Excel e per comprenderne il significato e la struttura durante la fase iniziale del progetto è stato fondamentale usare alcune funzionalità di base di Excel come il conteggio e la visualizzazione dei dati in grafici.

4.1.5 Modelli di Intelligenza Artificiale

Per svolgere il forecasting sono stati scelti alcuni modelli diversi, in modo da avere un benchmark che possa aiutare nella decisione del modello con la prestazione migliore. I modelli di machine learning coinvolti sono Random Forest e XGBoost, più un modello di deep learning, ovvero LSTM. Sono stati scelti questi modelli sia per la loro facilità di utilizzo, che per il loro basso costo computazionale. Il benchmark è stato svolto costruendo grafici per confrontare i risultati e calcolando le metriche di valutazione per ogni modello.

4.2 SCADA

Il sistema SCADA (Supervisory Control and Data Acquisition) è un insieme di strumenti e di programmi usati con il fine di raccogliere dati reali e di salvarli in una base di dati. Nel contesto di questo progetto, sono stati resi disponibili dati raccolti dai pozzi della sorgente e dal serbatoio principale. Le misurazioni sono effettuate ogni cinque minuti per un arco di tempo di alcuni anni. I dati di interesse sono quindi delle time series indicizzate con data e ora.

4.2.1 File di Input

I dati ottenuti dai sensori sono in formato Excel e ogni file contiene tutte le misure svolte in una giornata, divise in più fogli di calcolo, uno per ogni pozzo, prendendo la forma vista in Figura 4.1. I pozzi sono numerati da 1 a 6 e poi da 9 a 14, più il 14B, per un totale di 13 pozzi. La tabella contiene dati da diversi sensori, ma nel dataset finale va messa solo la parte utile in questo momento ed è composta da:

- **Portata pompe**: ogni pompa è dotata di un misuratore di portata che rileva lo scorrimento dell'acqua lungo la sezione della pompa in litri al secondo (l/s).
- **Corrente pompe**: ciascuna pompa viene attivata tramite un invertitore che ne consente l'attivazione graduale aumentando la corrente in Ampere (A), o equivalentemente aumentando la frequenza in Hertz (Hz). Questa è la grandezza che va minimizzata nel problema di ottimizzazione.

	A	C	D	E	F	G
1	POZZO 1 CAPTAZIONE					
2						
3						
4						
5						
6						
7		02/05/2021 00:05	02/05/2021 00:10	02/05/2021 00:15	02/05/2021 00:20	02/05/2021 00:25
8	Livello Pozzo (m s.l.m.)	155,020	155,040	155,036	155,045	155,050
9	Portata Mandata Pompa 1 (l/s)	121,600	121,775	122,100	122,125	122,475
10	Portata Mandata Pompa 2 (l/s)	0,000	0,000	0,000	0,000	0,000
11	Volume totalizzato Pompa 1 (mc)	2184120,000	2184157,000	2184194,000	2184230,000	2184267,000
12	Volume totalizzato Pompa 2 (mc)	2746953,000	2746953,000	2746953,000	2746953,000	2746953,000
13	Temperatura Acqua (C)	11,449	11,438	11,456	11,483	11,460
14	Conducibilità (uS/cm)	615,500	613,750	612,000	612,250	613,750
15	Corrente Assorbita Pompa 1 (A)	21,438	21,454	21,421	21,405	21,376
16	Corrente Assorbita Pompa 2 (A)	0,000	0,000	0,000	0,000	0,000

POZZO 1 | POZZO 2 | POZZO 3 | POZZO 4 | POZZO 5 | POZZO 6 | POZZO 9 | POZZO 10 | POZZO 11 | POZZO 12 | POZZO 13 | POZZO 14 | PO

Figura 4.1: Esempio di file input.

- **Livello acqua pozzi**: i livelli dell'acqua nei pozzi indicano quanta acqua rimane nel pozzo e influenzano direttamente la portata delle pompe: meno acqua c'è nel pozzo e minore sarà la portata della pompa a parità di corrente. Misurato in metri (m).
- **Livello acqua serbatoio**: il livello dell'acqua nel serbatoio principale indica quanta acqua è disponibile per la soddisfazione della domanda. È misurato in metri (m) e deve rimanere entro un certo limite, opportunamente definito.
- **Portata uscita serbatoio**: indica l'acqua in uscita dal serbatoio e rappresenta la richiesta idrica totale che dev'essere sempre soddisfatta. Misurata in litri al secondo (l/s).

Tutti i dati reali sono stati recuperati dai sistemi di archiviazione gestiti da VIVA Servizi S.p.A.

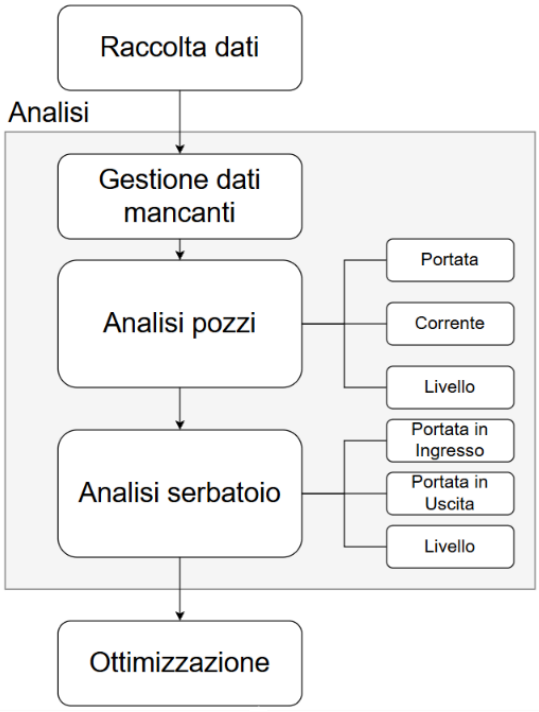


Figura 4.2: Metodologia per l' ottimizzazione energetica.

4.3 Metodologia per l'ottimizzazione energetica

Il progetto è costituito da 3 fasi principali (Fig. 4.2):

- **Raccolta dei dati**: i dati che sono stati forniti per lo svolgimento di questo lavoro sono stati selezionati e riadattati per gli scopi del progetto tramite uno script Python. Il risultato ottenuto rappresenta l'input per il passo successivo.
- **Analisi dei dati**: una corretta analisi è fondamentale per comprendere la natura del sistema e di tutti i fattori che lo influenzano. Le conoscenze ottenute in questa fase sono state cruciali durante lo sviluppo dell'ultimo step.
- **Ottimizzazione**: la parte finale del progetto consiste nella costruzione di un sistema di ottimizzazione che prende un input e lo elabora in modo da poter prevedere la richiesta idrica futura e in base alle condizioni del sistema attivare i pozzi secondo un certo ordine, minimizzando il consumo di corrente elettrica e soddisfacendo la richiesta. Infine, è stato creato un benchmark dei modelli di machine learning scelti per valutarne le prestazioni e i punti di forza.

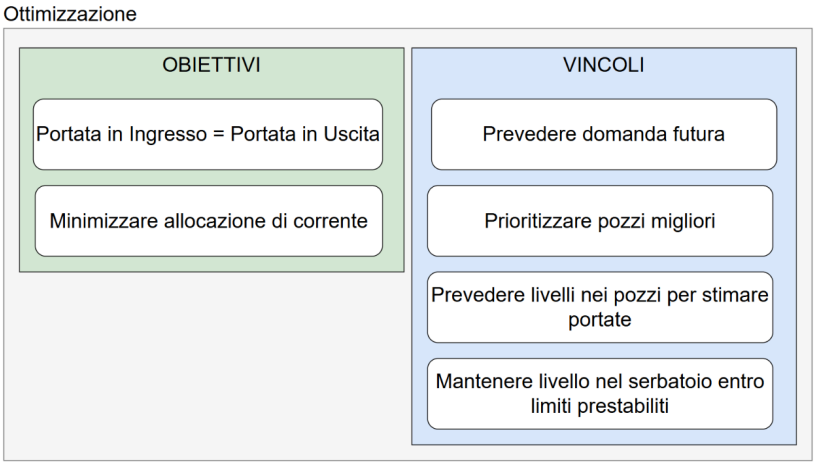


Figura 4.3: Struttura del problema di ottimizzazione: obiettivi e vincoli operativi.

La suddivisione in raccolta dati, analisi dati e ottimizzazione è dovuta alla propedeuticità di ogni passo rispetto al successivo. L'acquisizione dei dati dai file di input consiste nella selezione dei parametri di interesse e dalla costruzione di un singolo dataset principale con dati eterogenei e di qualità variabile. Questi dati sono stati analizzati a fondo e sulla base dei risultati si è sviluppato il modello data-driven di ottimizzazione.

Per via della presenza di valori mancanti nelle time series del dataset, il passo preliminare dell'analisi è stata proprio la loro gestione. Gli elementi dell'acquedotto di interesse per questo progetto sono i pozzi e il serbatoio, che sono caratterizzati da grandezze fisiche riportate nel dataset. Ogni pozzo contiene delle pompe, che vengono attivate per portare acqua verso il serbatoio e quindi verso la rete di distribuzione. Controllando i valori di corrente elettrica assegnati alle pompe si può controllare la portata delle pompe e di conseguenza la portata in ingresso al serbatoio. I valori storici sono definiti dalla gestione manuale delle pompe, attivate in base alla quantità d'acqua richiesta. Studiando la storia del sistema è stato possibile comprenderne il funzionamento e, a partire da queste conoscenze, si è progettato un metodo per la sua ottimizzazione.

L'obiettivo principale del modello di ottimizzazione (Fig. 4.3) consiste nel garantire il soddisfacimento della domanda idrica minimizzando il consumo di energia elettrica. Il processo di ottimizzazione si basa sulla previsione della domanda futura, ottenuta mediante un modello di intelligenza artificiale addestrato sui dati storici. La domanda idrica prevista, espressa in termini di portata in uscita, rappresenta il fabbisogno che il sistema deve soddisfare e determina, di conseguenza, la portata complessiva da prelevare dai pozzi.

La selezione dei pozzi viene fatta in modo da favorire quelli che storicamente sono stati utilizzati maggiormente, poiché strutturalmente migliori. La portata prodotta da ogni pompa dipende dalla corrente fornita alla stessa e dal livello d'acqua nel pozzo.

Per stimare la portata di una pompa bisogna considerare che il livello d'acqua è variabile e dipende da quanta acqua viene estratta dal pozzo, quindi si prevede il livello futuro in modo da avere risultati più accurati ad ogni timestep. L'ottimizzazione viene infatti svolta a una certa cadenza oraria definita nei parametri del modello.

Per far sì che il livello d'acqua nel serbatoio sia sufficiente a soddisfare richieste d'acqua più sostanziose, esso deve rimanere entro un certo livello prestabilito: se il livello va oltre le soglie minime e massime si deve modulare la distribuzione in modo da evitare il rischio di insufficienze e di sovraccarichi.

Capitolo 5

Progettazione e Sviluppo

In questo capitolo verrà descritto in modo dettagliato il lavoro svolto, inclusi il codice Python e i grafici generati.
Lo scopo del progetto è quello di creare una base operativa di un sistema di ottimizzazione delle risorse idriche della sorgente Gorgovivo utilizzando un approccio data-driven con metodi di machine learning, che possa minimizzare il consumo energetico dovuto all'attivazione dei pozzi. Nelle prossime sezioni viene spiegata l'implementazione delle tre fasi di raccolta, analisi e ottimizzazione in quest'ordine.

5.1 Raccolta dei dati

5.1.1 Lettura file

I dati messi a disposizione per questo progetto sono misurazioni giornaliere, effettuate ogni cinque minuti, dei sensori posti nei pozzi e nel serbatoio. In formato Excel, compongono una tabella la cui prima colonna contiene i nomi degli attributi e le successive colonne contengono i valori misurati. Ogni file Excel contiene un foglio di calcolo separato per ogni pozzo e per il serbatoio. Il primo script Python estrae dalla cartella di lavoro i file Excel uno ad uno ed elabora ogni foglio, producendo un file .csv per ogni pozzo contenente tutte le misurazioni.

Le librerie *glob* e *os* permettono di cercare nella cartella di lavoro tutti i file che iniziano per "Dati_Pozzi_", che vengono poi ordinati per data e inseriti in una lista. Viene poi creata la cartella in cui andrà inserito l'output generato. Il dizionario *well_data* conterrà i dati dei pozzi.

```
file_list = glob(os.path.join(data_dir, "raw data", "Dati_Pozzi_*.xlsx"))
file_list_sorted = sorted(file_list, key=extract_date_from_filename)
file_list_filtered = file_list_sorted

print(f"Found {len(file_list_filtered)} files to process (sorted by date).")
output_csv_dir = os.path.join(data_dir, "csv_output ok")
if not os.path.exists(output_csv_dir):
    os.makedirs(output_csv_dir)

well_data = defaultdict(list)
```

Per ordinare i file dei pozzi si cerca una corrispondenza ai nomi dei file e si estraggono l'anno il mese e il giorno. La struttura del nome di un file è di questo tipo:
Dati_Pozzi_<giorno>_<mese>_<anno>.xlsx

```
def extract_date_from_filename(filename):
    m = re.search(r"Dati_Pozzi_(\d+)_(\d+)_(\d+)\.xlsx", os.path.basename(
    filename))
    if m:
        day, month, year = m.groups()
        try:
            return datetime.datetime(int(year), int(month), int(day))
        except Exception:
            return datetime.datetime.min
    return datetime.datetime.min
```

Per ogni pozzo nella lista si estraggono i dati con la funzione *process_ excel_ file()* e vengono inseriti in *well_ data* usando come *key* il nome del pozzo.

```
for file in file_list_filtered:
    file_wells = process_excel_file(file)
    for well, df in file_wells.items():
        well_data[well].append(df)
```

Questa funzione prende il percorso di un file Excel e itera i fogli di calcolo nel file. Per ogni foglio viene chiamata la funzione *process_ excel_ sheet()* e i dati estratti vengono inseriti nel dizionario *sheet_ dict*. Le eccezioni vengono gestite col costrutto *try-except* e riportando l'errore nella console. La funzione restituisce il dizionario appena costruito.

```
def process_excel_file(file_path):
    print(f"\nProcessing file: {file_path}")
    try:
        xls = pd.ExcelFile(file_path)
        sheet_dict = {}
        for sheet in xls.sheet_names:
            df_sheet = process_excel_sheet(xls, sheet, file_path)
            if df_sheet is not None:
                sheet_dict[sheet] = df_sheet
        return sheet_dict
    except Exception as e:
        print(f"Error processing file {file_path}: {e}")
        return {}
```

Questa funzione legge i dati presenti in un foglio Excel e li inserisce in un DataFrame. Le colonne del foglio successive alla prima rappresentano i *timestamp*, mentre le colonne del DataFrame devono rappresentare i parametri della time series, quindi è trasporre la matrice del DataFrame. Per leggere correttamente le date si tentano diversi formati. Al termine della funzione viene chiamata *_clean_frame()* passando il DataFrame come parametro.

```
def process_excel_sheet(xls, sheet, file_path):
    print(f"  Processing sheet: {sheet}")
    try:
        df = pd.read_excel(xls, sheet_name=sheet, skiprows=6, header=0)
        if df.empty:
            print(f"    Sheet {sheet} is empty, skipping.")
            return None

        df.rename(columns={df.columns[0]: "Parameter"}, inplace=True)

        # Skip hidden column
        if 'PORTATE BRIGLIE' not in sheet:
            df.drop(df.columns[1], axis=1, inplace=True)

        timestamp_cols = df.columns[1:]
        new_cols = ["Parameter"]
        for col in timestamp_cols:
            col_str = str(col).strip()
            dt = None
            try:
                dt = pd.to_datetime(col_str, format="%Y-%m-%d %H:%M:%S", errors=
    "raise")
            except ValueError:
                try:
                    dt = pd.to_datetime(col_str, format="%Y-%m-%d %H:%M", errors
    ="raise")
                except ValueError:
                    dt = pd.to_datetime(col_str, errors="coerce")
            new_cols.append(dt)
        df.columns = new_cols

        df_transposed = df.set_index("Parameter").transpose()
        df_transposed["well_or_reservoir"] = sheet
        df_transposed["source_file"] = os.path.basename(file_path)

        return _clean_frame(df_transposed)

    except Exception as e:
        print(f"Error processing sheet {sheet} in file {file_path}: {e}")
        return None

```

L'ultima pulizia del DataFrame consiste nel rimuovere eventuali duplicati e nell'assicurarsi che tutti i timestamp siano univoci ed equidistanti. La colonna dei timestamp viene chiamata *Time*.

```
def _clean_frame(df):
    df = df.copy()

    # Remove duplicate columns before concat
    if df.columns.has_duplicates:
        df = df.loc[:, ~df.columns.duplicated(keep="first")]

    # Ensure the index is datetime and unique
    df.index = pd.to_datetime(df.index, errors="coerce")
    df.index = df.index.floor("s")
    df = df[~df.index.duplicated(keep="first")]
    df.index.name = "Time"
    return df
```

Per ogni pozzo nel dizionario *well_ dict* si uniscono tutti i relativi DataFrame e si crea un file csv, quindi al termine dell'esecuzione si ottengono tanti file csv quanti sono i fogli, ovvero per ogni pozzo più il serbatoio.

```
for well, dfs in well_data.items():
    if not dfs:
        continue

    cleaned_dfs = []
    for df in dfs:
        cleaned_dfs.append(_clean_frame(df))

    merged = pd.concat(cleaned_dfs, axis=0, join="outer", sort=False)
    merged = _clean_frame(merged)

    csv_filename = os.path.join(output_csv_dir, f"{well}.csv")

    if not os.path.exists(csv_filename):
        merged.to_csv(csv_filename, mode="w", index=True)
        print(f"Created CSV for well {well}")
    else:
        merged.to_csv(csv_filename, mode="a", header=False, index=True)
        print(f"Appended data for well {well} to {csv_filename}")
```

5.1.2 Costruzione Dataset

Il seguente script prende i file appena generati e li unisce per generare un unico file per il Dataset che verrà utilizzato nei prossimi step.

La cartella di input è la stessa dell'output dello script precedente.

```
    INPUT_FOLDER = "csv_output ok"
```

La lista *well_files* contiene i file di input che corrispondono ai pozzi.

```
well_files = glob(os.path.join(INPUT_FOLDER, "POZZO*.csv"))
```

La lista viene ordinata per assicurarsi che il pozzo 14B segua il pozzo 14.

```
def sort_key(f):
    base = os.path.basename(f)
    num_part = re.search(r'(\d+)', base)
    val = int(num_part.group(1)) if num_part else 0
    if 'B' in base.upper():
        val += 0.5
    return val

well_files.sort(key=sort_key)
```

Per ogni file si estrae il nome del pozzo dal percorso e si crea un DataFrame, impostando la colonna *Time* come indice e rimuovendo eventuali timestamp vuoti.

```
all_dfs = []

for f in well_files:
    try:
        filename = os.path.basename(f)
        wid = filename.replace("POZZO ", "").replace(".csv",
       "").strip()
        print(f"Processing Well {wid}...")

        df = pd.read_csv(f)

        if 'Time' in df.columns:
            df['Time'] = pd.to_datetime(df['Time'], errors='coerce')
            df.dropna(subset=['Time'], inplace=True)
            df.set_index('Time', inplace=True)
        else:
            continue
```

I nomi delle colonne vengono standardizzate per semplificare la lettura. La struttura è di questo tipo:
<Parametro>_W<pozzo>_P<pompa>
Dove i possibili parametri sono: Livello dell'acqua nel pozzo (*Level*), Portata della pompa (*Flow*), Corrente della pompa (*Amp*). Le unità di misura sono rispettivamente: metri sul livello del mare (m a.s.l.), litri al secondo (l/s), ampere (A).
Data la diversa struttura dei file è necessario specificare i casi dei pozzi 13 e 14B. Il pozzo 13 è l'unico ad avere 3 pompe, mentre il pozzo 14B ha 6 pompe ma possono venire considerate come una pompa unica.

```
1     rename_map = {}
2     for c in df.columns:
3         if '13' in f:
4             if "Livello Pozzo (m s.l.m.)" in c: rename_map[c] =
                  f"Level_W{wid}"
5             if "Portata Mandata Pompa 1" in c: rename_map[c] =
                  f"Flow_W{wid}_P1"
6             if "Portata Mandata Pompa 2" in c: rename_map[c] =
                  f"Flow_W{wid}_P2"
7             if "Portata Mandata Pompa 3" in c: rename_map[c] =
                  f"Flow_W{wid}_P3"
8             if "Corrente Assorbita Pompa 1" in c: rename_map[c] =
                  f"Amp_W{wid}_P1"
9             if "Corrente Assorbita Pompa 2" in c: rename_map[c] =
                  f"Amp_W{wid}_P2"
10            if "Corrente Assorbita Pompa 3" in c: rename_map[c] =
                  f"Amp_W{wid}_P3"
11        elif '14B' in f:
12            if "Livello Pozzo (m)" in c: rename_map[c] =
                  f"Level_W{wid}"
13            if "Portata Mandata Totale (l/s)" in c: rename_map[c]
                  = f"Flow_W{wid}_P0"
14            if "Corrente Assorbita Pompa 1" in c: rename_map[c] =
                  f"Amp_W{wid}_P1"
15            if "Corrente Assorbita Pompa 2" in c: rename_map[c] =
                  f"Amp_W{wid}_P2"
16            if "Corrente Assorbita Pompa 3" in c: rename_map[c] =
                  f"Amp_W{wid}_P3"
17            if "Corrente Assorbita Pompa 4" in c: rename_map[c] =
                  f"Amp_W{wid}_P4"
18            if "Corrente Assorbita Pompa 5" in c: rename_map[c] =
                  f"Amp_W{wid}_P5"
19            if "Corrente Assorbita Pompa 6" in c: rename_map[c] =
                  f"Amp_W{wid}_P6"
20        else:
21            if "Livello Pozzo (m s.l.m.)" in c: rename_map[c] =
                  f"Level_W{wid}"
22            if "Portata Mandata Pompa 1" in c: rename_map[c] =
                  f"Flow_W{wid}_P1"
23            if "Portata Mandata Pompa 2" in c: rename_map[c] =
                  f"Flow_W{wid}_P2"
24            if "Corrente Assorbita Pompa 1" in c: rename_map[c] =
                  f"Amp_W{wid}_P1"
25            if "Corrente Assorbita Pompa 2" in c: rename_map[c] =
                  f"Amp_W{wid}_P2"
26
27    df = df.rename(columns=rename_map)
28    df = df[list(rename_map.values())]
```

Tutti i dati vengono convertiti in unità numeriche. Per considerare il pozzo 14B come una pompa unica si fa la somma delle correnti di tutte le pompe.

```
    for c in df.columns:
        clean_str = df[c].astype(str).str.replace(",", ".")
        df[c] = pd.to_numeric(clean_str, errors='coerce')
        df[c] = df[c].clip(lower=0) # Pumps cannot have negative flow/amps

    df = df[~df.index.duplicated(keep='first')]
    all_dfs.append(df)
    if '14B' in f:
        df["Amp_W14B_P0"] = df.filter(regex=r"^Amp+_W14B+_P\d$").sum(axis=1)
        df.drop(columns=['Amp_W14B_P1', 'Amp_W14B_P2', 'Amp_W14B_P3',
                'Amp_W14B_P4', 'Amp_W14B_P5', 'Amp_W14B_P6' ], inplace=True)

    except Exception as e:
        print(f"Error reading {f}: {e}")

    if not all_dfs:
        print("No Well Data found. Check your INPUT_FOLDER path.")
        return
```

Il dataset finale viene costruito concatenando tutti i DataFrame dei pozzi. A questo punto si aggiungono anche i dati del serbatoio. In modo analogo a prima, si standardizzano i nomi dei parametri: *Tank_Level* per il livello d'acqua nel serbatoio e *Tank_Outflow* per la portata in uscita. Le unità di misura sono: millimetri (successivamente convertiti in metri) e litri al secondo. Si convertono le unità in numeri e si unisce il DataFrame a quello principale.

```
    master = pd.concat(all_dfs, axis=1, sort=True)

    external_data = [
        ("LIVELLO BRIGLIE*.csv", "Tank_Level", "Livello Briglia Monte
        (mm)", 1000.0),
        ("PORTATE BRIGLIE*.csv", "Tank_Outflow", "Portata Briglia Valle
        (l/s)", 1.0)
    ]

    for file_pattern, col_name, raw_col, scale in external_data:
        files = glob(os.path.join(INPUT_FOLDER, file_pattern))
        if files:
            print(f"Processing {col_name} from
                {os.path.basename(files[0])}...")
            ldf = pd.read_csv(files[0])
            ldf['Time'] = pd.to_datetime(ldf['Time'], errors='coerce')
            ldf.dropna(subset=['Time'], inplace=True)
            ldf.set_index('Time', inplace=True)

            # Remove duplicate timestamps to prevent Reindexing errors
            ldf = ldf[~ldf.index.duplicated(keep='first')]

            if raw_col in ldf.columns:
                # Clean strings and convert to numeric
                val_series =
                pd.to_numeric(ldf[raw_col].astype(str).str.replace(",",
                    "."), errors='coerce')

                temp_df = (val_series / scale).to_frame(name=col_name)
                master = master.join(temp_df, how='left')

        else:
             print(f" {file_pattern} not found.")
```

Infine si rimuovono le righe completamente vuote e si genera l'output. Questo file csv contiene tutti i dati utili all'analisi; il prossimo passo è la gestione dei valori mancanti.

```
master.dropna(how='all', inplace=True)

output = "MASTER_DATASET_NAN.csv"
master.to_csv(output)
```

5.2 Analisi dei dati

In questa sezione viene analizzato il dataset principale con grafici e rapporti. Per semplicità non verranno mostrati i grafici di tutti i pozzi, ma soltanto gli esempi più interessanti.

5.2.1 Gestione dei dati mancanti

Per svolgere un'analisi corretta di una time series è fondamentale che essa sia continua. Controllare la mancanza di dati è utile anche per verificare la qualità dei dati e per sapere se dovesse essere necessario porre delle modifiche al metodo di raccolta dati o, nel caso non fosse possibile, escludere dall'analisi la porzione di dati più sporca.

Il dataset viene caricato e reindicizzato per rilevare porzioni di dati mancanti dovute all'assenza di file per certi periodi di tempo. La cartella *plots* contiene tutti i grafici generati durante l'analisi ed è suddivisa in sottocartelle; i grafici di questa sezione verranno inseriti in *plots/nan*.

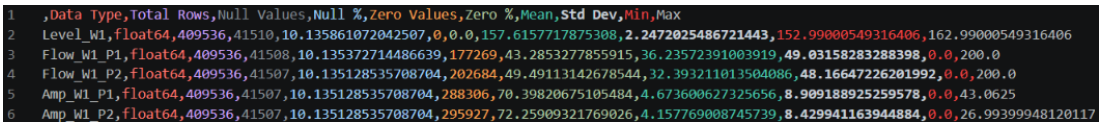
```
# Load the master dataset
df = pd.read_csv("MASTER_DATASET_NAN.csv", parse_dates=['Time'], index_col='Time')

# Reindex to detect intermittent missing data
full_index = pd.date_range(
    start=df.index.min(),
    end=df.index.max(),
    freq="5min",
    name="Time"
)
df = df.reindex(full_index)

if not os.path.exists("plots"):
    os.makedirs("plots")

if not os.path.exists("plots/nan"):
    os.makedirs("plots/nan")

plt.style.use('seaborn-v0_8-whitegrid')
sns.set_context("paper", font_scale=1.2)
```



	Data Type	Total Rows	Null Values	Null %	Zero Values	Zero %	Mean	Std Dev	Min	Max
Level_W1	float64	409536	41510	10.135861072042507	0	0.0	157.6157717875308	2.2472025486721443	152.99000549316406	162.99000549316406
Flow_W1_P1	float64	409536	41508	10.135372714486639	177269	43.2853277855915	36.23572391003919	49.03158283288398	0.0	200.0
Flow_W1_P2	float64	409536	41507	10.135128535708704	202684	49.49113142678544	32.393211013504086	48.16647226201992	0.0	200.0
Amp_W1_P1	float64	409536	41507	10.135128535708704	288306	70.39820675105484	4.673600627325656	8.909188925259578	0.0	43.0625
Amp_W1_P2	float64	409536	41507	10.135128535708704	295927	72.25909321769026	4.157769008745739	8.429941163944884	0.0	26.99399948120117

Figura 5.1: Porzione del report sulle variabili.

Il primo rapporto generato (Fig. 5.1) riguarda le proprietà delle variabili del dataset. Contiene alcune informazioni di base che caratterizzano le colonne del dataset, come il tipo di dato, il numero di righe, la quantità di valori nulli, la media, i massimi e i minimi.

```
summary = pd.DataFrame({
    'Data Type': df.dtypes,
    'Total Rows': len(df),
    'Null Values': df.isnull().sum(),
    'Null %': (df.isnull().sum() / len(df)) * 100,
    'Zero Values': (df == 0).sum(),
    'Zero %': ((df == 0).sum() / len(df)) * 100,
    'Mean': df.mean(),
    'Std Dev': df.std(),
    'Min': df.min(),
    'Max': df.max()
})

summary.to_csv("plots/nan/variable_summary_report.csv")
```

Per facilitare la comprensione, viene generato un grafico a barre sulla percentuale di valori nulli presenti in ogni colonna (Fig. 5.2).

```
perc_threshold = 12
plt.figure(figsize=(12, 6))
summary['Null %'].sort_values(ascending=False).plot(kind='bar', color='salmon')
plt.title('Missing Data % per Sensor', fontsize=14)
plt.ylabel('Percentage (%)')
plt.axhline(y=perc_threshold, color='r', linestyle='--', label=f'{perc_threshold
    }% Threshold')
plt.legend()
plt.tight_layout()
plt.savefig('plots/nan/missing_data_report.png', dpi=300)
plt.show()
```

Osservando il grafico si osserva che il pozzo 14B ha una grande quantità di dati mancanti, che potrebbe essere dovuto a errori di misurazione o problemi nel recupero dei dati dai sistemi di archiviazione. I pozzi 9, 10 e 14 hanno circa il 30% di valori mancanti, mentre il resto si tiene intorno al 10%. Di seguito vengono

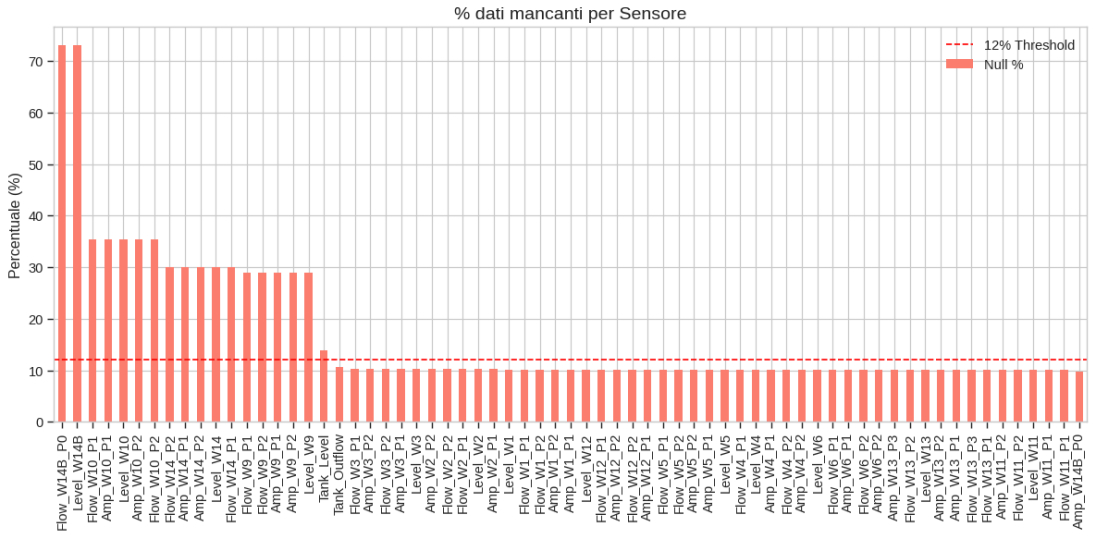


Figura 5.2: Percentuale NaN per ogni parametro.

visualizzate le colonne con le percentuali più critiche, in modo da comprendere meglio la natura dei dati. I valori in blu sono i valori validi e i valori mancanti sono evidenziati in arancione e posti a -1 (Fig. 5.3, 5.4, 5.5, 5.6).

```
nan_summary = summary.copy()
nan_summary = nan_summary.where(nan_summary['Null %'] > perc_threshold)
nan_summary = nan_summary.dropna()

for col in df.columns:
    if col in nan_summary.index:
        print(f"Plotting {col} nan values")
        nan_col = df[col].copy()
        # Missing values are set to -1
        nan_col = np.where(nan_col.isna(), -1, nan_col)
        plt.figure(figsize=(8,4))
        sns.lineplot(x=df.index, y=nan_col, color='darkorange', label= 'Missing data')
        sns.lineplot(x=df.index, y=df[col], color='dodgerblue', label= 'Valid data')
        plt.title(f'{col}', fontsize=14)
        plt.ylabel(f'{col}', fontsize=14)
        plt.tight_layout()
        plt.savefig(f'plots/nan/nan_{col}', dpi=300)
        plt.show()
```

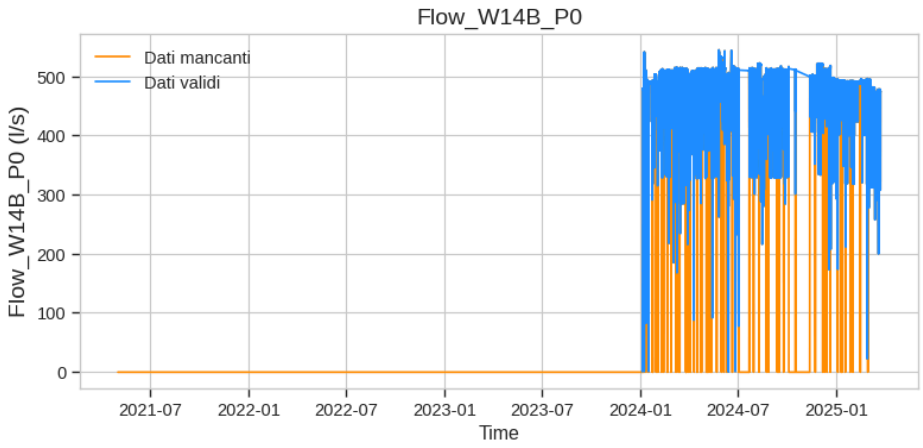


Figura 5.3: Dati validi e mancanti per la Portata del Pozzo 14B.

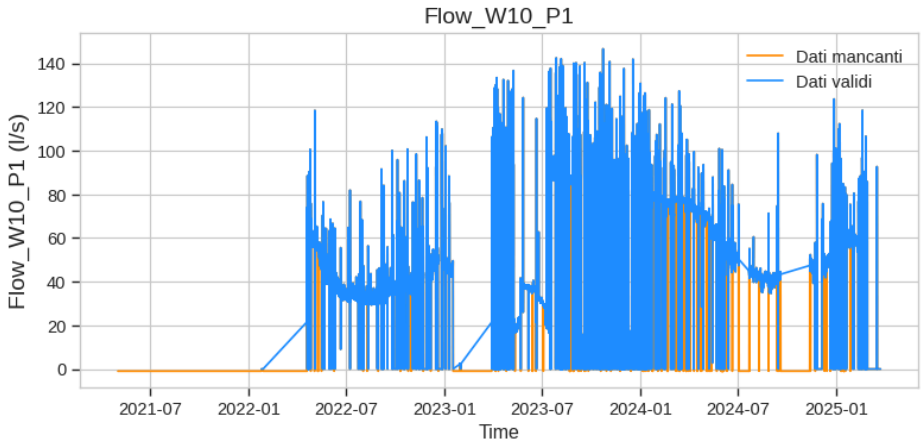


Figura 5.4: Dati validi e mancanti per la Portata della Pompa 1 del Pozzo 10.

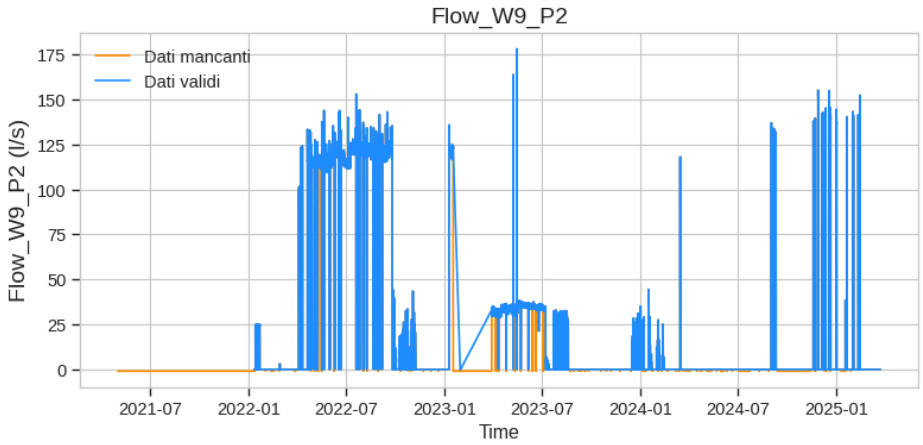


Figura 5.5: Dati validi e mancanti per la Portata della Pompa 2 del Pozzo 9.

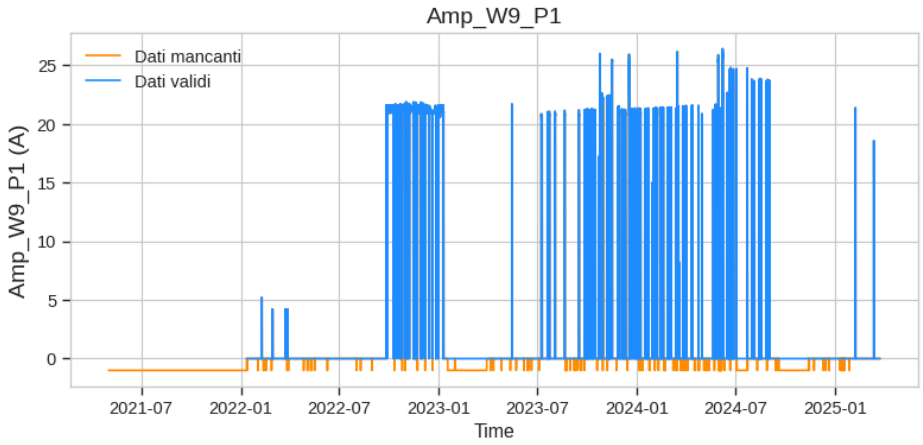


Figura 5.6: Dati validi e mancanti per la Portata della Pompa 1 del Pozzo 9.

Il pozzo 14B di fatto non ha dati dal 2021 al 2024 ma visto che non è il caso per gli altri pozzi, è possibile supporre che ci sia stato un problema nel caricamento dei dati dai sistemi di archiviazione.

Osservando i pozzi 9 e 10 si può dedurre che ci sono dei problemi nelle misurazioni dal 2021 al 2022, quindi si può valutare di escludere quel lasso di tempo nell'analisi, ma in questo caso è stato conservato. Durante le time series ci sono dei NaN intermittenti, ma si notano aree in comune tra tutti grafici, ovvero tra Gennaio e Marzo 2023 e tra Settembre e Novembre 2024. Questo può significare che i sensori fossero in stand-by in quei periodi oppure che i dati non sono stati estratti correttamente dai sistemi di archiviazione.

	Data Type	Total Rows	Null Values	Null %	Zero Values	Zero %	Mean	Std Dev	Min	Max
Level_W1	float64	409536	0	0.0	0	0.0	157.6157717875308	2.2472025486721443	152.99000549316406	162.99000549316406
Flow_W1_P1	float64	409536	0	0.0	189560	46.28652914517893	36.23572391003919	49.03158283288398	0.0	200.0
Flow_W1_P2	float64	409536	0	0.0	220966	53.955207844975774	32.393211013504086	48.16647226201992	0.0	200.0
Amp_W1_P1	float64	409536	0	0.0	318240	77.70745428973277	4.673600627325656	8.909188925259578	0.0	43.0625
Amp_W1_P2	float64	409536	0	0.0	331323	80.90204524144397	4.157769008745739	8.429941163944884	0.0	26.99399948120117

Figura 5.7: Porzione del report sulle variabili aggiornato.

Per colmare i valori mancanti nelle serie temporali, si è scelto di adottare l'interpolazione lineare, che consente di stimare ciascun valore mancante sulla base delle osservazioni disponibili nelle vicinanze temporali. Nel caso in cui rimanessero dei NaN, si occupano tutti gli spazi vuoti con degli zeri.

```
# Handle missing data
df = df.interpolate()
df = df.fillna(0)
```

Il rapporto sulle variabili viene aggiornato per verificare che non ci siano altri NaN nelle time series (Fig. 5.7).

```
updated_summary = summary.copy()
updated_summary['Null Values'] = df.isnull().sum()
updated_summary['Null %'] = (df.isnull().sum() / len(df)) * 100
updated_summary['Zero Values'] = (df == 0).sum()
updated_summary['Zero %'] = ((df == 0).sum() / len(df)) * 100
updated_summary.to_csv("plots/updated_summary_report.csv")

if updated_summary['Null %'].all() == 0:
    print("No nan values left")
```

Al termine dell'operazione non rimangono valori nulli, quindi il dataset è pronto per l'analisi completa.

```
output = "MASTER_DATASET_FINAL.csv"
df.to_csv(output)
print(f"Dataset Shape: {df.shape}")
print(f"Variables present: {list(df.columns)}")
```

5.2.2 Analisi Dati Esplorativa

L'EDA è suddivisa per tipo di dato. La prima parte è dedicata ai pozzi individuali, la seconda al serbatoio e l'ultima parte riguarda i dati dei pozzi aggregati.

5.2.2.1 Pozzi individuali

Lo scopo di analizzare i pozzi individualmente è quello di poter osservare come sono state utilizzate le pompe nel tempo, quali sono le eventuali relazioni tra i parametri dei pozzi e per capire secondo quale criterio si dovrebbero selezionare i pozzi da attivare durante l'ottimizzazione in base al loro utilizzo storico.

Prima di creare grafici, il sistema viene mappato prendendo dai nomi delle colonne gli *id* dei pozzi e associandovi quelli delle pompe, in questo modo si semplifica di molto la selezione di colonne per i grafici.

```
if not os.path.exists("plots/pumps"):
os.makedirs("plots/pumps")

plt.style.use('seaborn-v0_8-whitegrid')
sns.set_context("paper", font_scale=1.2)
file_path = 'MASTER_DATASET_FINAL.csv'

if os.path.exists(file_path):
    print("Loading MASTER_DATASET_FINAL.csv\n")
    df = pd.read_csv(file_path, parse_dates=['Time'], index_col='Time')

    # Map Wells and Pumps
    system_map = {}
    for col in df.columns:
        match = re.search(r'Flow_W([A-Za-z0-9]+)_P(\d+)', col)
        if match:
            well_id, pump_id = match.groups()
            if well_id not in system_map: system_map[well_id] = []
            system_map[well_id].append(pump_id)
```

I primi grafici (Fig. 5.8, 5.9, 5.10) mostrano l'evolversi della portata e della corrente di ogni pompa, per ogni pozzo. Quindi si ottiene un grafico per ogni pozzo, dove vengono visualizzati: in blu la portata della pompa 1, in arancione la corrente della pompa 1, in verde la portata della pompa 2 e in rosso la corrente della pompa 2. Vanno specificati i casi particolari dei pozzi 13 e 14B per una corretta visualizzazione. Si possono impostare diverse date per regolare il lasso di tempo da visualizzare, in questo caso viene visualizzato tutto il dataset.

```
# ---Plot Flow and Amp for each Pump per Well---
for w, pumps in system_map.items():

    start_date = '2021-05-02 00:00:00'
    end_date = '2025-03-24 00:00:00'
    mask = (df.index >= start_date) & (df.index <= end_date)

    small_df = df.loc[mask]
    print(f'Plotting Well_{w} Pump Behavior')
    if '14B' in w:
      plt.figure(figsize=(12,6))
      sns.lineplot(x=small_df.index, y=small_df[f'Flow_W{w}_P0'],
                        color='dodgerblue', label='Flow_P0')
      sns.lineplot(x=small_df.index, y=small_df[f'Amp_W{w}_P0'],
                        color='darkorange', label='Amp_P0')
      plt.ylabel('Flow (l/s), Amp (A)')
      plt.title(f'W{w} Pump Behavior', fontsize=14)
      plt.tight_layout()
      plt.savefig(f'plots/pumps/W{w}.png', dpi=300)
      plt.show()
    else:
      plt.figure(figsize=(12,6))
      sns.lineplot(x=small_df.index, y=small_df[f'Flow_W{w}_P1'],
                        color='dodgerblue', label='Flow_P1')
      sns.lineplot(x=small_df.index, y=small_df[f'Flow_W{w}_P2'],
                        color='green', label='Flow_P2')
      if len(pumps) == 3:
          sns.lineplot(x=small_df.index, y=small_df[f'Flow_W{w}_P3'],
                        color="#d26ff4", label='Flow_P3')

      sns.lineplot(x=small_df.index, y=small_df[f'Amp_W{w}_P1'],
                        color='darkorange', label='Amp_P1')
      sns.lineplot(x=small_df.index, y=small_df[f'Amp_W{w}_P2'], color='red',
                        label='Amp_P2')
      if len(pumps) == 3:
          sns.lineplot(x=small_df.index, y=small_df[f'Amp_W{w}_P3'],
                        color="#2323f3", label='Amp_P3')

      plt.ylabel('Flow (l/s), Amp (A)')
      plt.title(f'W{w} Pump Behavior', fontsize=14)
      plt.tight_layout()
      plt.savefig(f'plots/pumps/W{w}.png', dpi=300)
      plt.show()
```

Dai grafici si possono notare caratteristiche simili per tutti i pozzi. Nella maggior parte dei casi le pompe all'interno di un pozzo vengono alternate: quando la pompa 1 è accesa, la pompa 2 è spenta e viceversa. I livelli di corrente usati sono lievemente variabili e maggiore la corrente, maggiore è la portata della pompa. Tutti i pozzi hanno punti di lavoro diversi, dovuti alle loro caratteristiche fisiche e nonostante la scarsità dei dati del pozzo 14B, questo è quello con la portata massima maggiore.

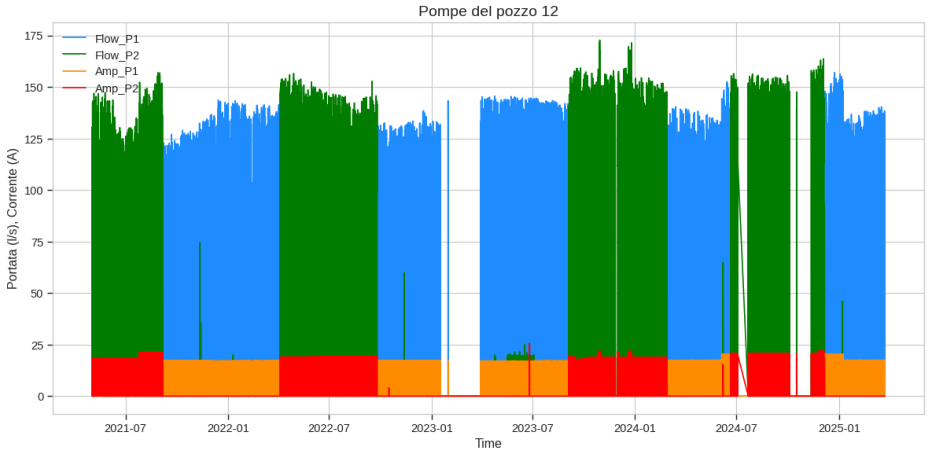


Figura 5.8: Portata e Corrente delle Pompe del Pozzo 12.

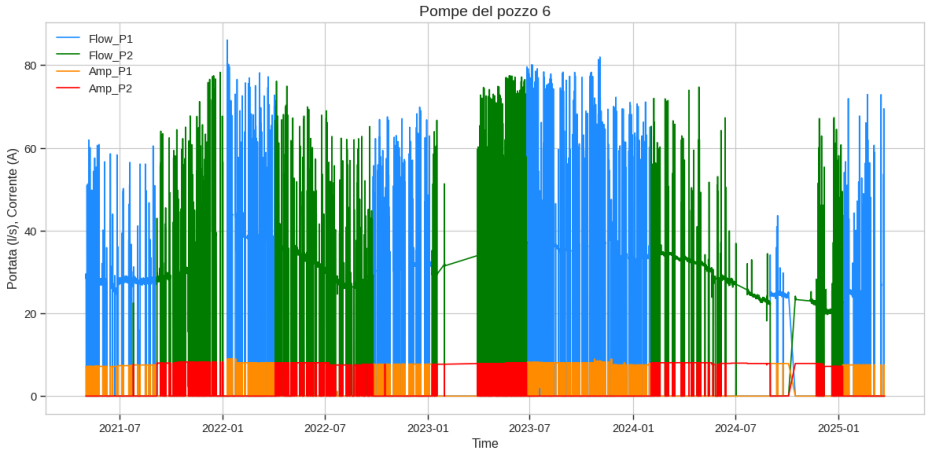


Figura 5.9: Portata e Corrente delle Pompe del Pozzo 6.

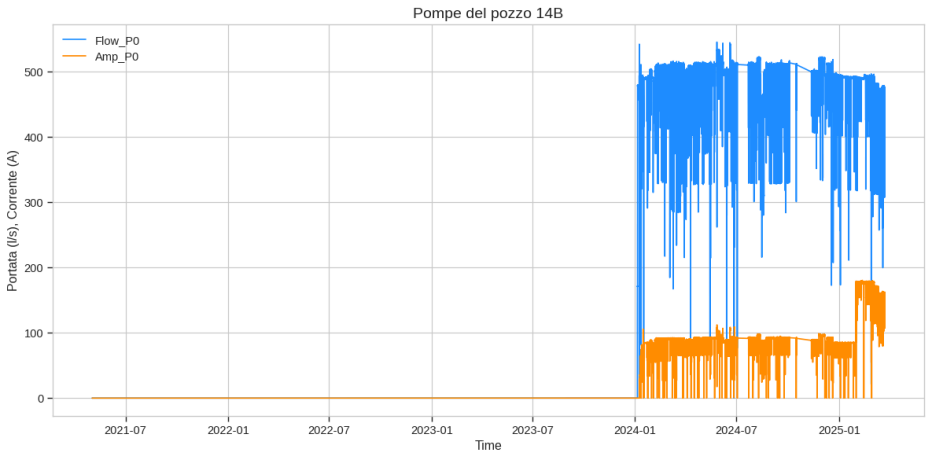


Figura 5.10: Portata e Corrente del Pozzo 14B.

Per comprendere la relazione tra corrente e portata si sono utilizzati Dynamic Time Warping (DTW) e correlazione. Prima di usare questi algoritmi le colonne della time series vanno normalizzate.

```
for c in df.columns:
    values = df[c].values
    values = values.reshape((len(values), 1))
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(values)
    normalized = scaler.transform(values)
    df[c] = normalized
```

DTW viene applicato con il metodo Sakoe-Chiba con finestre a tre dimensioni diverse, per trovare il risultato migliore con maggiore efficienza. La distanza viene calcolata tra portata e corrente per ogni pompa. Distanze minori significano maggiore similitudine (e quindi dipendenza) tra le due time series. Il risultato viene salvato in un file csv (Fig. 5.11).

```
window1 = 10
window2 = 100
window3 = 1000
distances1 = {}
distances2 = {}
distances3 = {}
for w, pumps in system_map.items():
    for p in pumps:
        pump_name = f"W{w}_P{p}"
        flow_col = f"Flow_{pump_name}"
        amp_col = f"Amp_{pump_name}"
        Flow = np.array(df[flow_col].values, dtype=np.double)
        Amp = np.array(df[amp_col].values, dtype=np.double)
        if Flow.any() > 0:
            distance1 = dtw.distance_fast(Flow, Amp,
                use_pruning=True, window=window1)
            distance2 = dtw.distance_fast(Flow, Amp,
                use_pruning=True, window=window2)
            distance3 = dtw.distance_fast(Flow, Amp,
                use_pruning=True, window=window3)
        else:
            distance1 = 0
            distance2 = 0
            distance3 = 0
        distances1[pump_name] = distance1
        distances2[pump_name] = distance2
        distances3[pump_name] = distance3
        print(f'DTW Distance (window: {window1}) (Flow-Amp) for
        {pump_name}: {distance1}')
        print(f'DTW Distance (window: {window2}) (Flow-Amp) for
        {pump_name}: {distance2}')
        print(f'DTW Distance (window: {window3}) (Flow-Amp) for
        {pump_name}: {distance3}\n')
profile_df[f'DTW Distance (Flow-Amp) window: {window1}'] = profile_df['Pump'].
    map(distances1)
profile_df[f'DTW Distance (Flow-Amp) window: {window2}'] = profile_df['Pump'].
    map(distances2)
profile_df[f'DTW Distance (Flow-Amp) window: {window3}'] = profile_df['Pump'].
    map(distances3)
profile_df.to_csv('plots/pumps/pump_profiles.csv')
```

```
DTW Distance (window: 10) (Flow-Amp) for W11_P1: 24.522449116977537
DTW Distance (window: 100) (Flow-Amp) for W11_P1: 24.085761035738006
DTW Distance (window: 1000) (Flow-Amp) for W11_P1: 23.791325423210914

DTW Distance (window: 10) (Flow-Amp) for W11_P2: 30.08702019426936
DTW Distance (window: 100) (Flow-Amp) for W11_P2: 29.89994828511237
DTW Distance (window: 1000) (Flow-Amp) for W11_P2: 29.835588992569253

DTW Distance (window: 10) (Flow-Amp) for W12_P1: 46.63381331987611
DTW Distance (window: 100) (Flow-Amp) for W12_P1: 46.184367521162294
DTW Distance (window: 1000) (Flow-Amp) for W12_P1: 46.022473641378625

DTW Distance (window: 10) (Flow-Amp) for W12_P2: 31.98553011654285
DTW Distance (window: 100) (Flow-Amp) for W12_P2: 31.302630268579016
DTW Distance (window: 1000) (Flow-Amp) for W12_P2: 30.761039956724435

DTW Distance (window: 10) (Flow-Amp) for W13_P1: 48.09975564464162
DTW Distance (window: 100) (Flow-Amp) for W13_P1: 47.98971305524339
DTW Distance (window: 1000) (Flow-Amp) for W13_P1: 47.559670625167165

DTW Distance (window: 10) (Flow-Amp) for W13_P2: 47.552318170083566
DTW Distance (window: 100) (Flow-Amp) for W13_P2: 47.04620420522977
DTW Distance (window: 1000) (Flow-Amp) for W13_P2: 45.82573294037275

DTW Distance (window: 10) (Flow-Amp) for W13_P3: 43.51724528094564
DTW Distance (window: 100) (Flow-Amp) for W13_P3: 43.067676764747205
DTW Distance (window: 1000) (Flow-Amp) for W13_P3: 41.70311978735314
```

Figura 5.11: Porzione del report sulle distanze di DTW per Portata-Corrente.

Le distanze sono relativamente basse in tutte le coppie portata-corrente, quindi si può concludere che seguono un andamento molto simile. Per dare ulteriori conferme si può analizzare il risultato del calcolo della correlazione con il coefficiente di Pearson.
La correlazione tra ogni coppia portata-corrente viene calcolata tramite la funzione *corr()* di Pandas e viene convertita in una *heatmap*, insieme formano un' unica matrice per una visualizzazione semplificata (Fig. 5.12). Valori vicini all'unità indicano alta correlazione lineare.

```python
corrs = {}
for w, pumps in system_map.items():
    for p in pumps:
        pump_name = f"W{w}_P{p}"
        flow_col = f"Flow_{pump_name}"
        amp_col = f"Amp_{pump_name}"
        if df[flow_col].any() > 0:
            corrs[pump_name] = df[flow_col].corr(df[amp_col])

profile_df['Correlation (Flow-Amp)'] = profile_df['Pump'].map(corrs)

# Convert to DataFrame (1 row)
corr_df = pd.DataFrame(corrs, index=["Flow vs Amp"])

plt.figure(figsize=(max(10, len(corrs)*0.6), 2.5))
sns.heatmap(corr_df, annot=True, fmt=".2f", cmap="coolwarm", center=0.5, vmin=0,
    vmax=1, linewidths=0.5)
plt.title("Flow-Amp Correlation for Each Pump")
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig("plots/pumps/Correlation_Flow_Amp_All_Pumps.png", dpi=300,
    bbox_inches="tight")
plt.show()
```

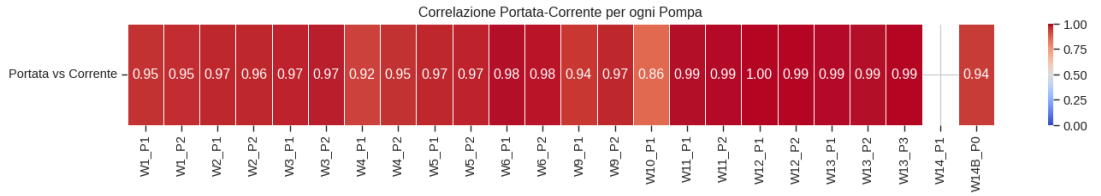


Figura 5.12: Correlazione Portata-Corrente per ciascuna Pompa di ogni Pozzo.

Anche in questo caso è evidente la correlazione tra le time series, avendo valori molto vicini a 1. Lo scarto può essere dovuto a inconsistenze tra i dati, a eventuali perdite nel sistema fisico o a ritardi tra variazioni di corrente e portata. Alla luce di questi risultati, si può affermare che quando viene innescata una corrente in una pompa, questa si attiva e apporta una modifica alla sua portata attuale quasi istantaneamente. Il pozzo 14 non ha un coefficiente di correlazione per via di insufficienza di dati.

Per progettare il sistema di ottimizzazione è utile sapere quanto viene usata ogni pompa all'interno di un pozzo e quali sono i pozzi più utilizzati. I seguenti grafici indicano il contributo di ogni pompa al totale di portata e corrente del proprio pozzo e il contributo di ogni pozzo al totale globale.

I pozzi vengono ordinati per nome e per ogni pozzo vengono generati due grafici a torta (uno per la portata e uno per la corrente), che vengono organizzati in una matrice. Ogni grafico rappresenta la portata (o corrente) totale di una pompa confrontata con le altre pompe del pozzo.

```
colors = sns.color_palette('Set3')

well_names = sorted(system_map.keys(), key=lambda x: int(re.sub('[^0-9]', '', x)
    ) if re.sub('[^0-9]', '', x) else x)
n_wells = len(well_names)
if n_wells == 0:
    print('No wells found for pie chart generation.')
else:
    ncols = min(3, n_wells)
    nrows = int(np.ceil(n_wells / ncols))
    fig_flow, axs_flow = plt.subplots(nrows=nrows, ncols=ncols,
                figsize=(5 * ncols, 4 * nrows))
    fig_amps, axs_amps = plt.subplots(nrows=nrows, ncols=ncols,
                figsize=(5 * ncols, 4 * nrows))
    if nrows * ncols == 1:
        axs_flow = np.array([axs_flow])
        axs_amps = np.array([axs_amps])
    else:
        axs_flow = np.array(axs_flow).reshape(-1)
        axs_amps = np.array(axs_amps).reshape(-1)
    for ax in axs_flow[n_wells:]:
        ax.axis('off')
    for ax in axs_amps[n_wells:]:
        ax.axis('off')
```

```
    for idx, w in enumerate(well_names):
        pumps = system_map[w]
        flow_data = {}
        amps_data = {}
        for p in pumps:
            pump_name = f"W{w}_P{p}"
            flow_col = f"Flow_{pump_name}"
            amp_col = f"Amp_{pump_name}"
            flow_sum = df[flow_col].sum()
            amp_sum = df[amp_col].sum()
            if flow_sum > 0:
                flow_data[pump_name] = flow_sum
            if amp_sum > 0:
                amps_data[pump_name] = amp_sum
        ax_flow = axs_flow[idx]
        ax_amps = axs_amps[idx]
        if flow_data:
            flow_sorted = {k: v for k, v in sorted(flow_data.items(),
                           key=lambda item: item[1])}
            ax_flow.pie(flow_sorted.values(),labels=flow_sorted.keys(),
            colors=colors[:len(flow_sorted)],autopct='%.0f%%')
            ax_flow.set_title(f'Flow Contribution-W{w}', fontsize=12)
        else:
            ax_flow.text(0.5, 0.5, 'No flow data', ha='center',va='center')
            ax_flow.axis('off')
        if amps_data:
            amps_sorted = {k: v for k, v in sorted(amps_data.items(),
                           key=lambda item: item[1])}
            ax_amps.pie(amps_sorted.values(),labels=amps_sorted.keys(),
                      colors=colors[:len(amps_sorted)], autopct="%.0f%%")
            ax_amps.set_title(f'Amp Contribution-W{w}', fontsize=12)
        else:
            ax_amps.text(0.5, 0.5, 'No amp data', ha='center', va='center')
            ax_amps.axis('off')
    fig_flow.suptitle('Total Flow Contribution by Well', fontsize=16,
                      fontweight='bold')
    fig_flow.tight_layout()
    fig_flow.savefig('plots/pumps/Pie_Flow_All_Wells.png', dpi=300)
    plt.show()
    fig_amps.suptitle('Total Amp Contribution by Well', fontsize=16,
                      fontweight='bold')
    fig_amps.tight_layout()
    fig_amps.savefig('plots/pumps/Pie_Amps_All_Wells.png', dpi=300)
    plt.show()
```

I grafici nelle Figure 5.13 e 5.14 mostrano che le pompe di ogni pozzo tendono a essere utilizzate in modo equo, questa cosa non è casuale in quanto così facendo l'usura delle pompe rimane bilanciata e l'efficienza del sistema è mantenuta più alta possibile. I pozzi 10 e 14 presentano un'insufficienza di dati per fare le stesse conclusioni ma si può dedurre che il loro comportamento sia analogo a quello degli altri.

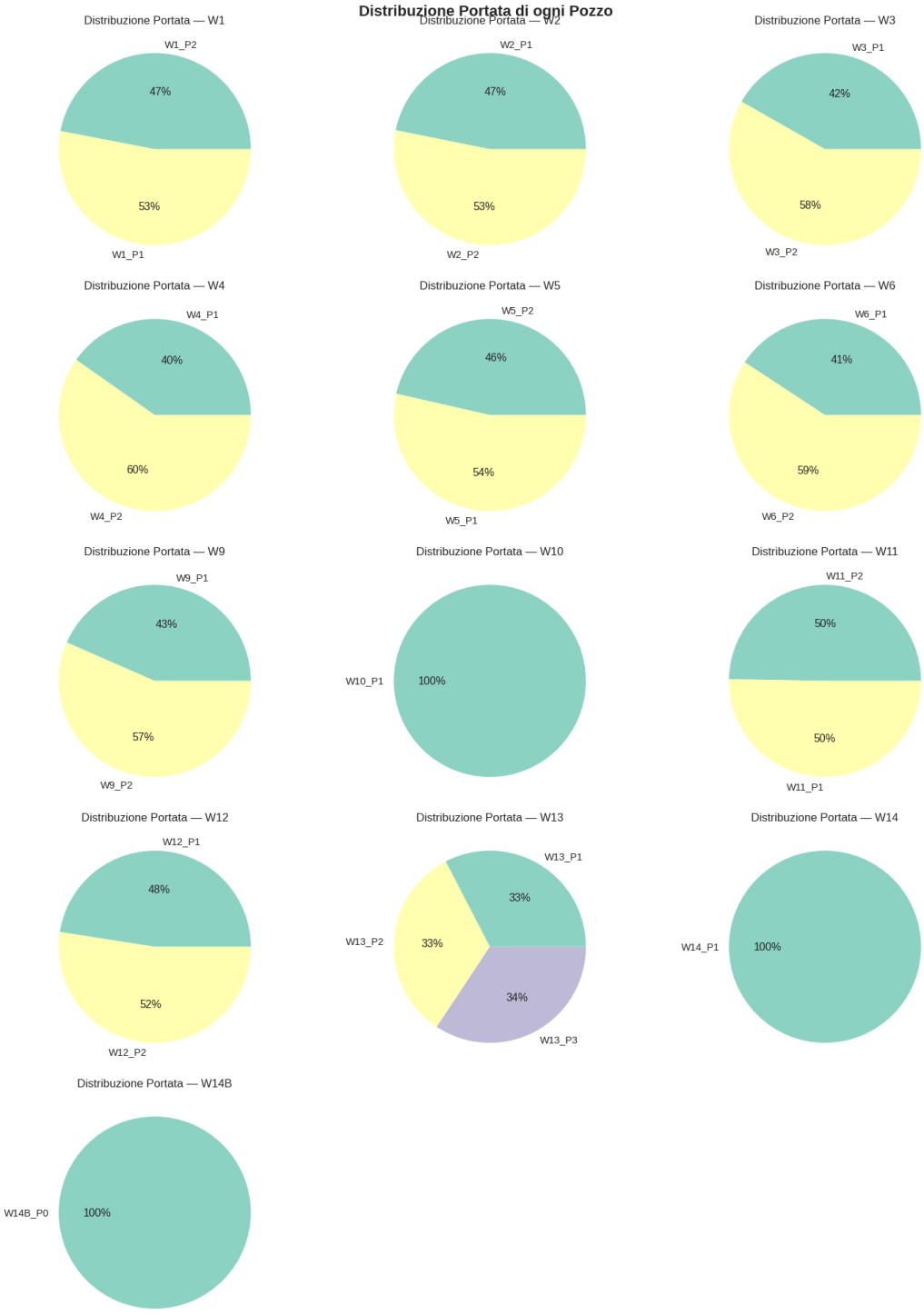


Figura 5.13: Contributo di ogni Pompa alla Portata complessiva di ciascun Pozzo.

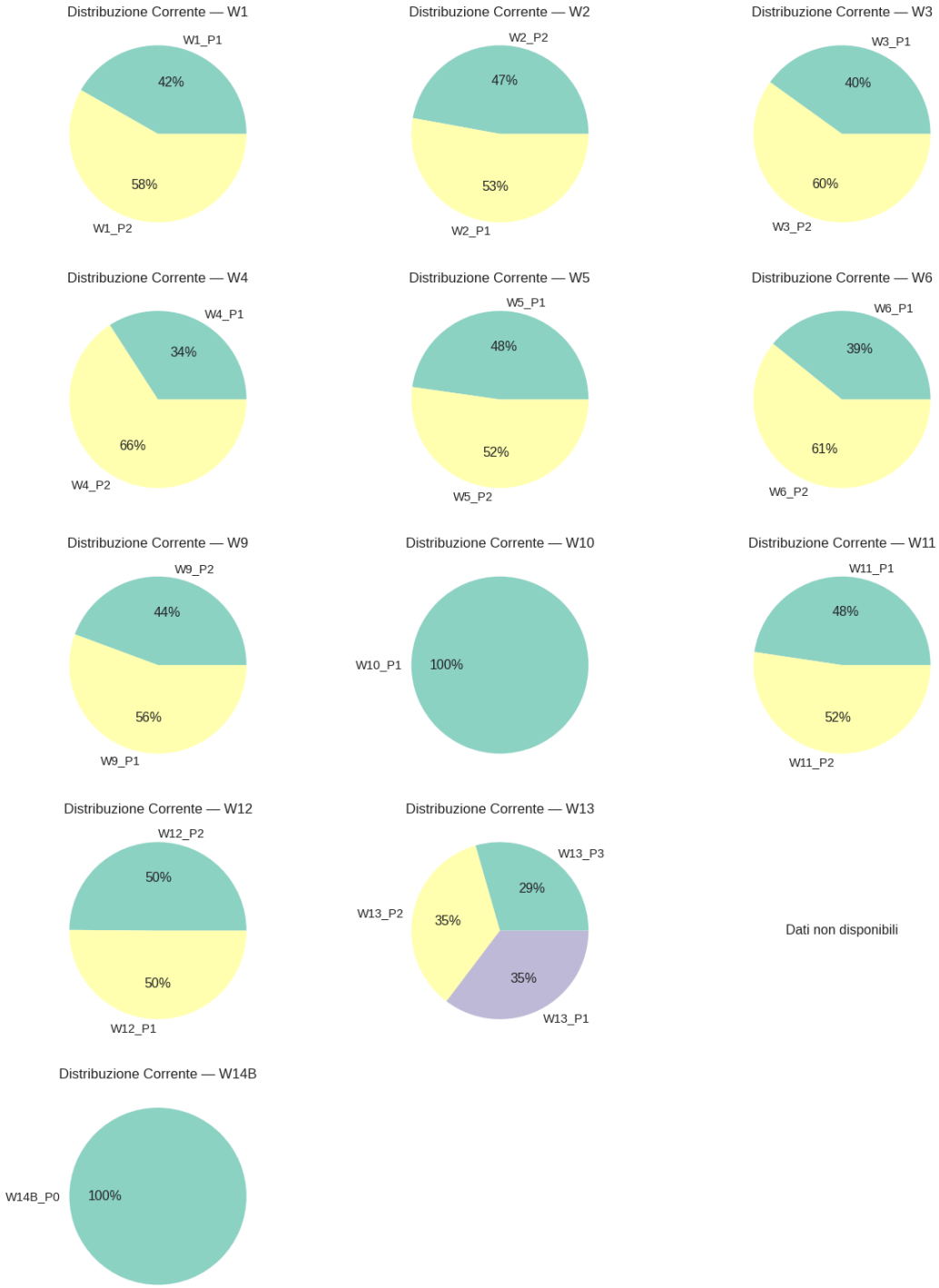


Figura 5.14: Contributo di ogni Pompa alla Corrente complessiva consumata dalle pompe di ciascun Pozzo.

Per visualizzare il contributo di ogni pozzo al totale vengono usati due grafici a torta (Fig 5.15, 5.16), con la somma di portate e correnti di ogni pozzo confrontate con il totale. Le sezioni dei grafici sono poi ordinate in base al contributo maggiore.

```
well_flow_totals = {}
well_amp_totals = {}
for w in well_names:
    flow_total = 0
    amp_total = 0
    for p in system_map[w]:
        pump_name = f"W{w}_P{p}"
        flow_total += df[f"Flow_{pump_name}"].sum()
        amp_total += df[f"Amp_{pump_name}"].sum()
    if flow_total > 0:
        well_flow_totals[f"W{w}"] = flow_total
    if amp_total > 0:
        well_amp_totals[f"W{w}"] = amp_total

if well_flow_totals:
    fig_well_flow, ax_well_flow = plt.subplots(figsize=(8, 6))
    sorted_flow = {k: v for k, v in sorted(well_flow_totals.items(),
                   key=lambda item: item[1])}
    ax_well_flow.pie(sorted_flow.values(),labels=sorted_flow.keys(),
                     colors=colors[:len(sorted_flow)],autopct='%.0f%%')
    ax_well_flow.set_title('Total Flow Distribution Across Wells',
                    fontsize=14)
    fig_well_flow.tight_layout()
    fig_well_flow.savefig('plots/pumps/Pie_Flow_Per_Well.png', dpi=300)
    plt.show()

if well_amp_totals:
    fig_well_amp, ax_well_amp = plt.subplots(figsize=(8, 6))
    sorted_amps = {k: v for k, v in sorted(well_amp_totals.items(),
                   key=lambda item: item[1])}
    ax_well_amp.pie(sorted_amps.values(),labels=sorted_amps.keys(),
                    colors=colors[:len(sorted_amps)], autopct='%.0f%%')
    ax_well_amp.set_title('Total Amp Distribution Across Wells',
                    fontsize=14)
    fig_well_amp.tight_layout()
    fig_well_amp.savefig('plots/pumps/Pie_Amps_Per_Well.png', dpi=300)
    plt.show()
```

Il pozzo più utilizzato è il 13, probabilmente per via delle sue caratteristiche fisiche e per il fatto che è dotato di 3 pompe a differenza degli altri. Ricordando che il pozzo 14B ha un'insufficienza di dati ma un'alta portata massima, è lecito pensare che il suo contributo sia in realtà molto maggiore.

Questa classifica verrà poi usata durante l'ottimizzazione per decidere l'ordine di attivazione dei pozzi.

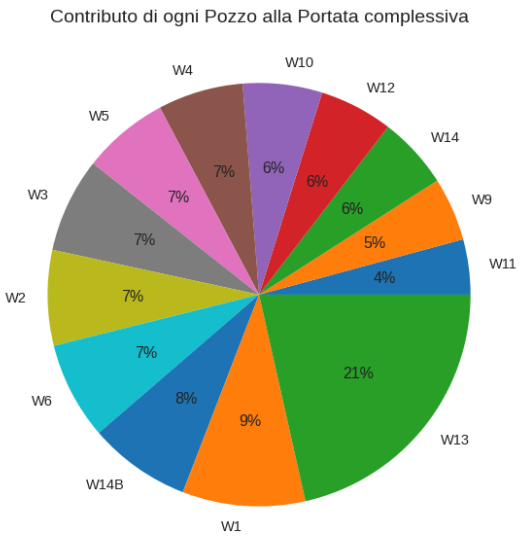


Figura 5.15: Contributo di ogni Pozzo alla Portata totale dell'intero sistema.

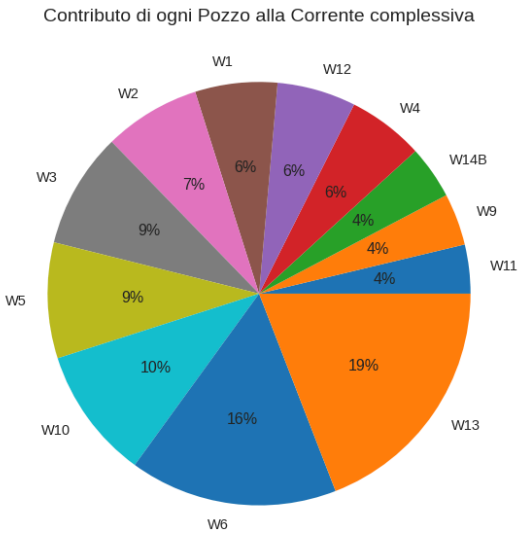


Figura 5.16: Contributo di ogni Pozzo al consumo di Corrente totale dell'intero sistema.

5.2.2.2 Livelli dei Pozzi

Il livello dell'acqua nei pozzi influenza la portata delle pompe, quindi è importante capire come si relazionano in modo da poter usare i livelli come parametro aggiuntivo per l'ottimizzazione.

La disponibilità d'acqua può dipendere dal periodo dell'anno, quindi visualizzare la stagionalità della time series può dare informazioni aggiuntive. Per avere una rappresentazione veritiera dell'andamento dei livelli si devono confrontare soltanto campioni presi quando la portata nel pozzo è nulla, altrimenti il livello risulterebbe influenzato dall'attività delle pompe. Per farlo, quindi, si ricampiona il DataFrame a frequenza mensile e si prende un campione del livello d'acqua per ogni mese, controllando che la portata in quel momento sia nulla. Per facilitare la lettura del grafico in Figura 5.17 sono stati inclusi soltanto quattro pozzi (1, 2, 3 e 13).

Nel periodo estivo i livelli tendono ad abbassarsi, il che significa che la disponibilità d'acqua è minore e invece durante il periodo invernale-primaverile i livelli sono più alti.

```
df['Month'] = df.index.month
print("\nPlotting monthly well levels")
zero_flow_samples = []
for w, pumps, in system_map.items():
    flow_cols = [f'Flow_W{w}_P{p}' for p in pumps if f'Flow_W{w}_P{p}' in
                 df.columns]
    if not flow_cols:
        continue
    mask = df[flow_cols].eq(0).all(axis=1)
    zero_df = df.loc[mask, [f'Level_W{w}', 'Month'] + flow_cols].copy()
    if not zero_df.empty:
        zero_df = zero_df.reset_index()
        zero_df = zero_df.groupby('Month', sort=True).first().reset_index()
        zero_df = zero_df.rename(columns={'Time': 'Sample_Time',
                                 f'Level_W{w}': 'Level'})
        zero_df['Well'] = w
        zero_flow_samples.append(zero_df[['Well', 'Month', 'Sample_Time',
                                 'Level']])
if zero_flow_samples:
    zero_flow_df = pd.concat(zero_flow_samples, ignore_index=True)

    plt.figure(figsize=(12, 6))
    sns.scatterplot(data=zero_flow_df, x='Month', y='Level', hue='Well',
                    style='Well', s=100)
    plt.title('Monthly Well Level Samples with All Pumps Zero Flow')
    plt.xlabel('Month (1-12)')
    plt.xticks(range(1, 13))
    plt.ylabel('Well Level (m)')
    plt.tight_layout()
    plt.savefig('plots/levels/Zero_Flow_Monthly_Well_Level_Samples.png',
                    dpi=300)
    plt.show()
```

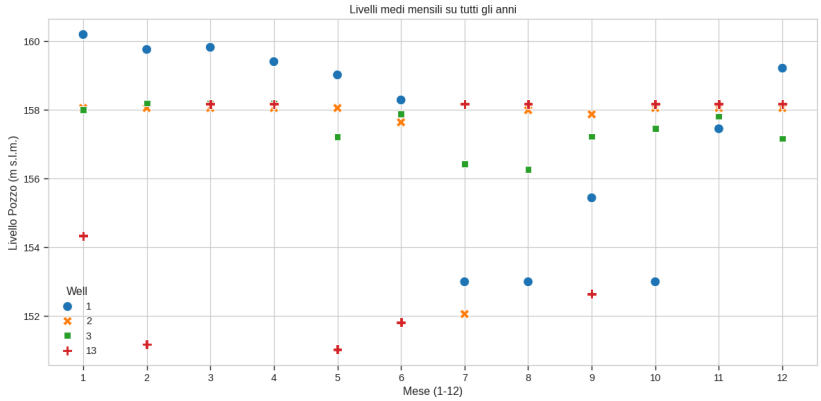


Figura 5.17: Livelli d'acqua nei pozzi mensili medi calcolati su tutti gli anni.

La relazione tra portata e livello può essere evidenziata visualizzando i loro comportamenti e usando DTW. Prima di ogni operazione, le colonne vengono normalizzate.

In Figura 5.18 la portata è in blu, mentre il livello è in verde. Osservando i risultati si nota che portata e livello si muovono in maniera opposta, fatto confermato dalle distanze di DTW molto elevate (Fig. 5.19). Questo significa che all'aumentare dell'attività delle pompe, ovvero della portata, il livello si abbassa e dato che il salto tra la superficie dell'acqua e la sommità del pozzo aumenta, la corrente necessaria a mantenere la stessa portata aumenta.

Il livello dell'acqua sarà un parametro importante durante l'ottimizzazione e conoscerne l'andamento aiuterà a fare decisioni migliori nella selezione delle pompe.

```
for w, pumps in system_map.items():
df[f"Well{w}_Flow"] = df.filter(regex=rf"^Flow+_W{w}+_P\d$").sum(axis=1)
df[f"Well{w}_Amp"] = df.filter(regex=rf"^Amp+_W{w}+_P\d$").sum(axis=1)

# ---Plot Flow and Amp for each Pump per Well---
for w, pumps in system_map.items():

start_date = '2021-05-02 00:05:00'
end_date = '2025-03-24 00:00:00'
mask = (df.index >= start_date) & (df.index <= end_date)

small_df = df.loc[mask]
print(f'Plotting Well_{w} water levels')

plt.figure(figsize=(12, 6))
sns.lineplot(x=small_df.index, y=small_df[f"Well{w}_Flow"], color='dodgerblue',
    label=f'Flow_W{w}')
sns.lineplot(x=small_df.index, y=small_df[f'Level_W{w}'], color='green', label=f
    'Level_W{w}')
plt.ylabel('Well Level (m a.s.l.)')
plt.title(f'W{w} Water Level', fontsize=14)
plt.tight_layout()
plt.savefig(f'plots/levels/W{w}.png', dpi=300)
plt.show()

well_profiles = []

# ---Dynamic Time Warping---
window1 = 10
window2 = 100
window3 = 1000
print("\nCalculating DTW Distance (Flow-Level) for each Well")
for w, pumps in system_map.items():
flow_col = f"Well{w}_Flow"
level_col = f"Level_W{w}"
Flow = np.array(df[flow_col].values, dtype=np.double)
Level = np.array(df[level_col].values, dtype=np.double)
if Flow.any() > 0:
    distance1 = dtw.distance_fast(Flow, Level, use_pruning=True, window=window1)
    distance2 = dtw.distance_fast(Flow, Level, use_pruning=True, window=window2)
    distance3 = dtw.distance_fast(Flow, Level, use_pruning=True, window=window3)
else:
    distance1 = 0
    distance2 = 0
    distance3 = 0

print(f'DTW Distance (window: {window1}) (Flow-Level) for {w}: {distance1}')
print(f'DTW Distance (window: {window2}) (Flow-Level) for {w}: {distance2}')
print(f'DTW Distance (window: {window3}) (Flow-Level) for {w}:{distance3}')
well_profiles.append({
            'Well': w,
            f'DTW Distance (Flow-Level) window: {window1}': distance1,
            f'DTW Distance (Flow-Level) window: {window2}': distance2,
            f'DTW Distance (Flow-Level) window: {window3}': distance3
        })

profile_df = pd.DataFrame(well_profiles)
profile_df.to_csv('plots/levels/well_profiles.csv')
```

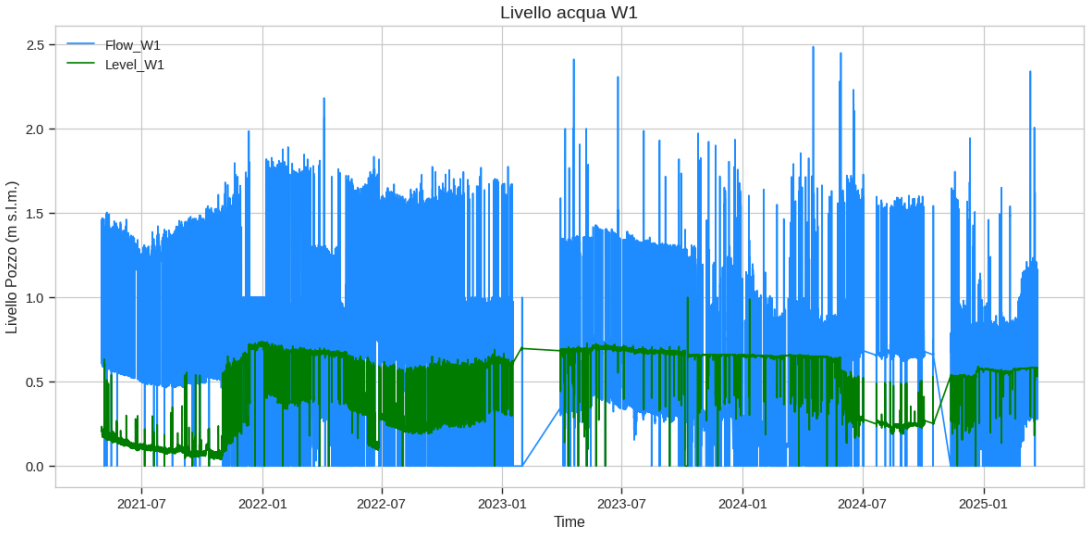


Figura 5.18: Livello-Portata Pozzo 1.

```
DTW Distance (window: 10) (Flow-Level) for 6: 333.5275835793933
DTW Distance (window: 100) (Flow-Level) for 6: 281.83594530172576
DTW Distance (window: 1000) (Flow-Level) for 6: 172.58839644003046

DTW Distance (window: 10) (Flow-Level) for 9: 469.3385358290436
DTW Distance (window: 100) (Flow-Level) for 9: 468.9850996932547
DTW Distance (window: 1000) (Flow-Level) for 9: 466.57429540947857

DTW Distance (window: 10) (Flow-Level) for 10: 413.53088594544937
DTW Distance (window: 100) (Flow-Level) for 10: 413.2446278298163
DTW Distance (window: 1000) (Flow-Level) for 10: 411.6173980055021

DTW Distance (window: 10) (Flow-Level) for 11: 411.48419788718576
DTW Distance (window: 100) (Flow-Level) for 11: 382.8966705325974
DTW Distance (window: 1000) (Flow-Level) for 11: 317.0210278741555

DTW Distance (window: 10) (Flow-Level) for 12: 413.2302793791649
DTW Distance (window: 100) (Flow-Level) for 12: 315.680217311519
DTW Distance (window: 1000) (Flow-Level) for 12: 270.8541330455054

DTW Distance (window: 10) (Flow-Level) for 13: 283.4406753797307
DTW Distance (window: 100) (Flow-Level) for 13: 278.7672606864995
DTW Distance (window: 1000) (Flow-Level) for 13: 253.96556506500917

DTW Distance (window: 10) (Flow-Level) for 14: 456.0943786147261
DTW Distance (window: 100) (Flow-Level) for 14: 455.9841518896023
DTW Distance (window: 1000) (Flow-Level) for 14: 454.97515922427255
```

Figura 5.19: Porzione del report sulle distanze di DTW per Livello-Portata.

5.2.2.3 Serbatoio

I due parametri che definiscono lo stato del serbatoio sono: livello dell'acqua (in metri) e portata in uscita (in litri al secondo). Il livello dell'acqua (Fig. 5.20) è un indicatore di cruciale importanza, poiché questo non deve scendere mai sotto una certa soglia, per garantire una riserva d'acqua sufficiente. La portata in uscita (Fig. 5.21) rappresenta la richiesta d'acqua totale, in quanto questa è l'acqua che entra nella rete di distribuzione dell'acquedotto.

Il livello viene mantenuto tra i 3.4 m e i 4 m ma sempre sopra i 3 m. La portata in uscita oscilla tra gli 800 l/s e i 1200 l/s.

Successivamente, si vuole trovare la relazione tra livello e portata, quindi si normalizzano (Fig. 5.22) e si calcolano correlazione (Fig. 5.23) e distanza di DTW (Fig. 5.24).

```
plt.figure(figsize=(12, 6))
sns.lineplot(x=df.index, y=df.Tank_Level, color="#059e44", label='Level')
plt.title('Tank Level and Outflow', fontsize=14, fontweight='bold')
plt.ylabel('Tank Level')
plt.xlabel('Date')
plt.tight_layout()
plt.savefig('plots/tank/Tank_Level.png', dpi=300)
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(x=df.index, y=df.Tank_Outflow, color='#109ee4', label='Outflow')
plt.title('Tank Level and Outflow', fontsize=14, fontweight='bold')
plt.ylabel('Tank Outflow')
plt.xlabel('Date')
plt.tight_layout()
plt.savefig('plots/tank/Tank_Outflow.png', dpi=300)
plt.show()
```

```
# ---CORRELATION---
print('\nCalculating Correlation (Pearson)')
plt.figure(figsize=(10, 4))
corr = df[['Tank_Level', 'Tank_Outflow']].corr()
sns.heatmap(corr[['Tank_Level']], annot=True, cmap='coolwarm', fmt='.2f',
            linewidths=0.5)
plt.title('Tank Level and Outflow Correlation')
plt.savefig('plots/tank/Correlation_Level_Outflow.png', dpi=300)
plt.show()
```

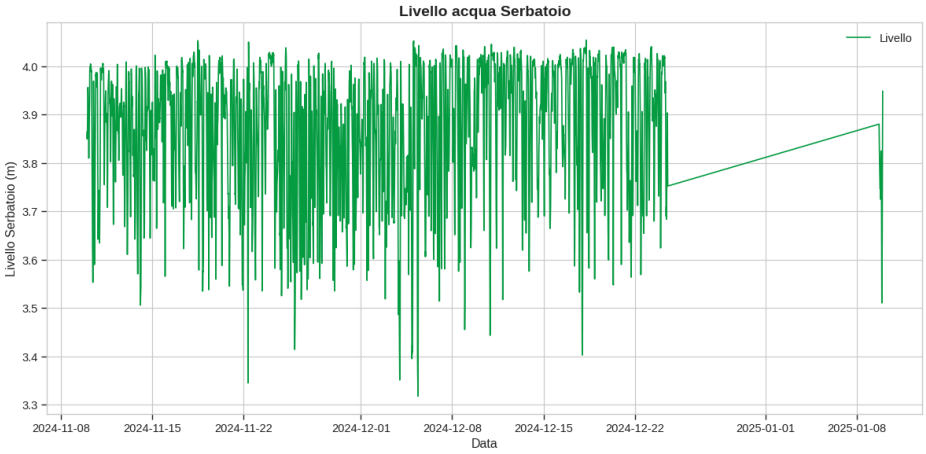


Figura 5.20: Livello Serbatoio.

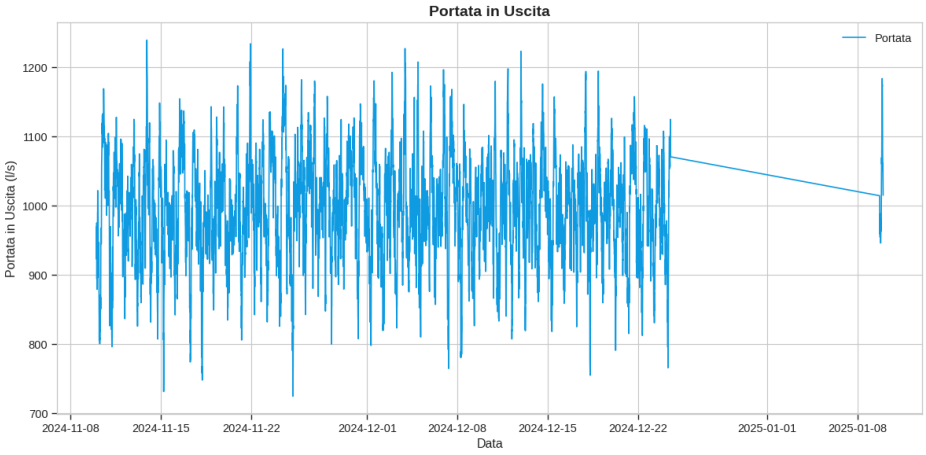


Figura 5.21: Portata in Uscita Serbatoio.

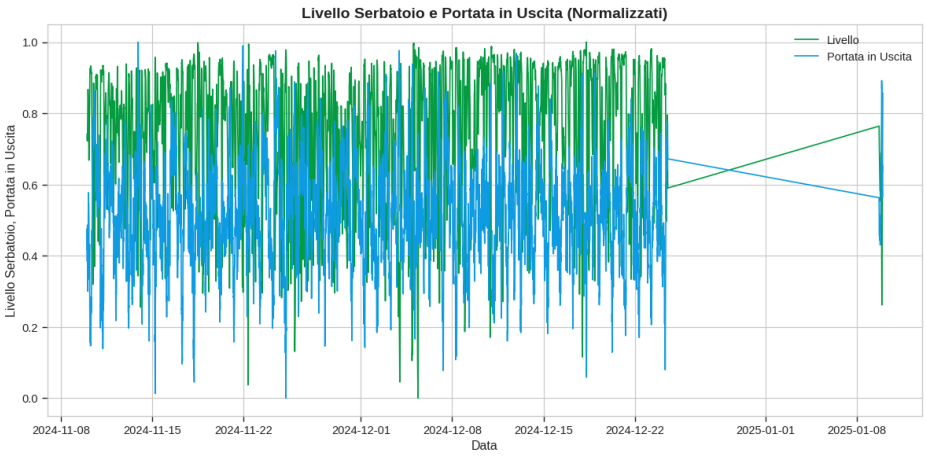


Figura 5.22: Livello e Portata in Uscita (Normalizzati).

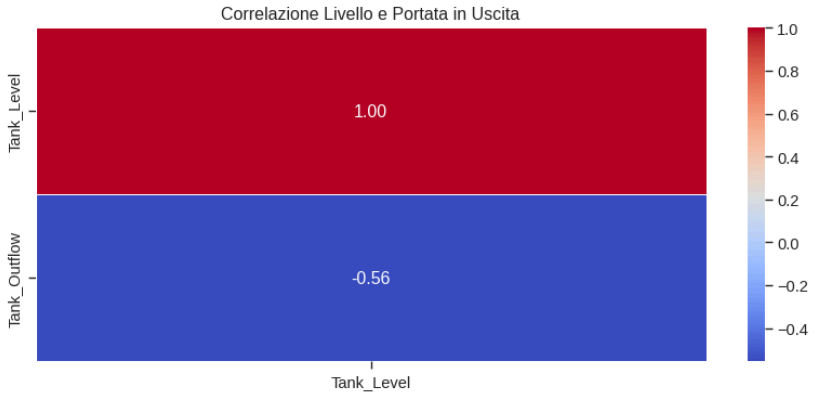


Figura 5.23: Correlazione Livello-Portata in Uscita (Serbatoio).

```
Calculating DTW Distance
DTW Distance between Tank Level and Outflow (window=1000): 10.011610663035722
DTW Distance between Tank Level and Outflow (window=10000): 9.709422558315804
Filtering time series
DTW Distance for Tank Level and Outflow (filtered) (window=1000): 9.539785197856062
DTW Distance for Tank Level and Outflow (filtered) (window=10000): 9.189993153621218
```

Figura 5.24: Distanze di DTW per Livello-Portata in Uscita (Serbatoio).

In questo caso il coefficiente di Pearson non fornisce risultati significativi, quindi è preferibile considerare soltanto la distanza di DTW.

Per cercare di ottenere risultati migliori riducendo il rumore dei dati, si applica un filtro di Hampel a entrambe le serie. La distanza DTW viene calcolata prima e dopo aver applicato il filtro. Il filtro migliora i risultati dell'algoritmo, che produce comunque distanze molto basse.

Da questo si può concludere che le due serie si muovono in maniera molto simile, ma non c'è linearità.

```
# ---DYNAMIC TIME WARPING---
print("\nCalculating DTW Distance")

dtw_profile = []

Level = np.array(df['Tank_Level'].values, dtype=np.double)
Outflow = np.array(df['Tank_Outflow'].values, dtype=np.double)

window1 = 1000
window2 = 10000
distance1 = dtw.distance_fast(Level, Outflow, use_pruning=True, window=1000)
distance2 = dtw.distance_fast(Level, Outflow, use_pruning=True, window=10000)

dtw_profile.append({
    'Parameters': 'Level-Outflow',
    f'DTW (window: {window1})': distance1,
    f'DTW (window: {window2})': distance2
})


print(f'DTW Distance between Tank Level and Outflow (window={window1}):
{distance1}')
print(f'DTW Distance between Tank Level and Outflow (window={window2}):
{distance2}')

# Apply Hampel filter
print('Filtering time series')
result_Tank_Level = hampel(df['Tank_Level'], window_size=72)
filtered_Tank_Level = result_Tank_Level.filtered_data
result_Outflow = hampel(df['Tank_Outflow'], window_size=72)
filtered_Outflow = result_Outflow.filtered_data

Level = np.array(filtered_Tank_Level.values, dtype=np.double)
Outflow = np.array(filtered_Outflow.values, dtype=np.double)

distance1_filtered = dtw.distance_fast(Level, Outflow, use_pruning=True, window=
    window1)
distance2_filtered = dtw.distance_fast(Level, Outflow, use_pruning=True, window=
    window2)

dtw_profile.append({
    'Parameters': 'Level-Outflow (filtered)',
    f'DTW (window: {window1})': distance1,
    f'DTW (window: {window2})': distance2
})
print(f'DTW Distance for Tank Level and Outflow (filtered) (window={window1}): {
    distance1_filtered}')
print(f'DTW Distance for Tank Level and Outflow (filtered) (window={window2}): {
    distance2_filtered}')

# Keep best results
if distance2_filtered < distance2:
    df['Tank_Level'] = filtered_Tank_Level.values
    df['Tank_Outflow'] = filtered_Outflow.values

dtw_df = pd.DataFrame(dtw_profile)
dtw_df.to_csv('plots/tank/dtw_tank_profile.csv')
```

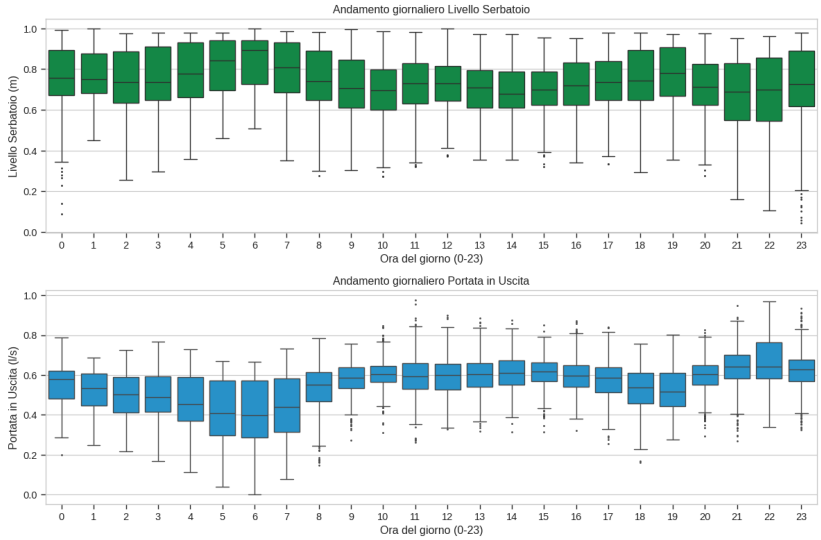


Figura 5.25: Andamento medio giornaliero della Portata in Uscita e del Livello d'acqua del Serbatoio.

Altri grafici d'interesse riguardano la stagionalità giornaliera dei parametri del serbatoio (Fig. 5.25). Analizzandoli si possono osservare le richieste della popolazione che definiscono il funzionamento di questo sistema. Si può osservare che i momenti in cui la richiesta idrica aumenta sono il pomeriggio e la sera, mentre di notte fino al mattino diminuisce. Al momento dati di questo tipo non sono inclusi nell'ottimizzazione ma in futuro si possono includere per migliorarne l'efficacia e rendendola in grado di rispondere alle abitudini umane e regolare la distribuzione d'acqua in base ad esse.

```
# ---DAILY SEASONALITY---
df['Hour'] = df.index.hour

print("\nPlotting daily Tank_Level and Tank_Outflow")
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(12,8))

sns.boxplot(x='Hour', y='Tank_Level', data=df, color="#059e44", fliersize=1, ax=
    ax[0])
ax[0].title.set_text('Diurnal Tank Level (Level by Hour of Day)')
ax[0].set_ylabel('Tank Level')
ax[0].set_xlabel('Hour of the Day (0-23)')

sns.boxplot(x='Hour', y='Tank_Outflow', data=df, color="#109ee4", fliersize=1,
    ax=ax[1])
ax[1].title.set_text('Diurnal Water Demand Profile (Outflow by Hour of Day)')
ax[1].set_ylabel('Outflow Rate')
ax[1].set_xlabel('Hour of the Day (0-23)')

plt.tight_layout()
plt.savefig('plots/tank/Daily_Tank.png', dpi=300)
plt.show()

df.drop(columns=['Hour'], inplace=True)
```

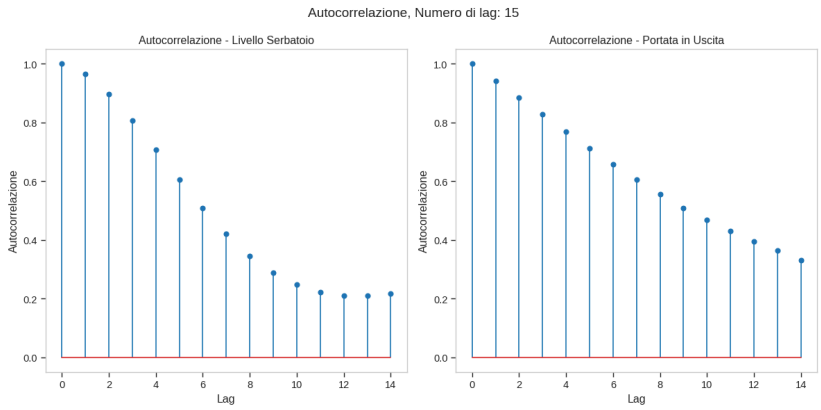


Figura 5.26: Autocorrelazione (Serbatoio).

Infine si calcola l'autocorrelazione di entrambe le time series (Fig. 5.26), parametro che può essere utile per certi modelli di forecasting.

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12,6))
n_lags = 15
lags = np.arange(0, n_lags)
Level = df['Tank_Level']
Outflow = df['Tank_Outflow']
Level_Autocorrs = [Level.autocorr(lag=lag) for lag in lags]
Outflow_Autocorrs = [Outflow.autocorr(lag=lag) for lag in lags]
ax[0].stem(lags, Level_Autocorrs)
ax[0].grid()
ax[0].title.set_text('Autocorrelation in Tank_Level')
ax[1].stem(lags, Outflow_Autocorrs)
ax[1].grid()
ax[1].title.set_text('Autocorrelation in Tank_Outflow')
fig.suptitle(f"Autocorrelation, Number of lags: {n_lags}")
plt.tight_layout()
plt.savefig('plots/tank/Autocorr_Tank', dpi=300)
plt.show()
```

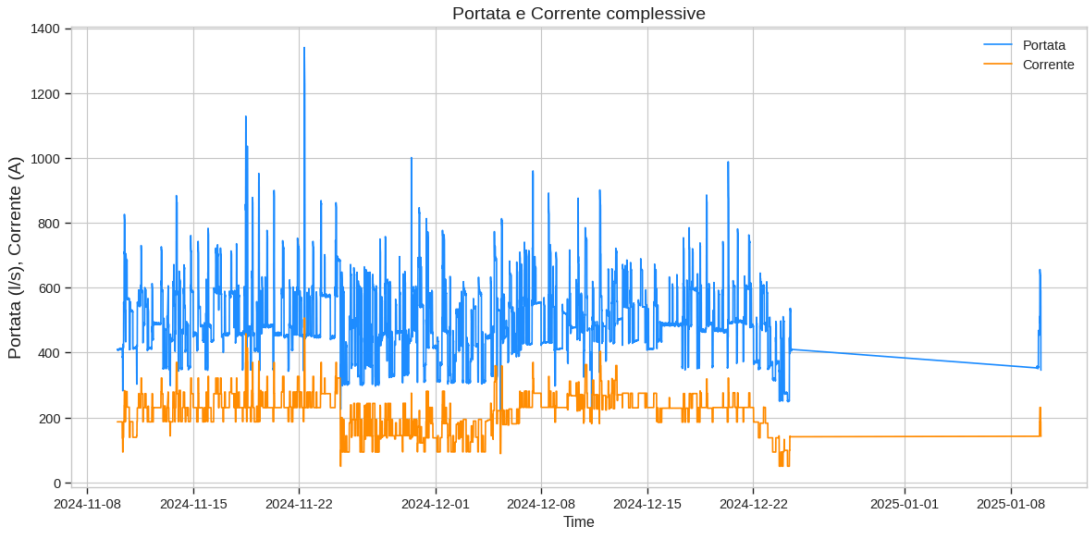


Figura 5.27: Dati dei Pozzi Aggregati.

5.2.2.4 Pozzi aggregati

In questa sezione vengono analizzati i dati dei pozzi aggregati, ovvero la somma delle portate e delle correnti di tutti i pozzi e vengono messi a confronto con i dati del serbatoio. I dati aggregati indicano portata totale e corrente totale, come mostrato in Figura 5.27, e vengono salvati in un dataset separato. La portata totale rappresenta la portata in entrata del serbatoio, il cui livello d'acqua interno dipende proprio da portata in entrata e portata in uscita. La corrente totale verrà utilizzata successivamente per calcolare il consumo energetico dovuto alla gestione manuale del sistema.

```
df = pd.read_csv(file_path, parse_dates=['Time'], index_col='Time')
    aggregated_df = pd.DataFrame()
    aggregated_df["Total_Flow"] = df.filter(regex=r"^Flow+_W\d+_P\d$").sum(axis
    =1)
    aggregated_df["Total_Amps"] = df.filter(regex=r"^Amp+_W\d+_P\d$").sum(axis
    =1)
    output = 'TOTAL_AGGREGATED_WELLS.csv'
    aggregated_df.to_csv(output)
```

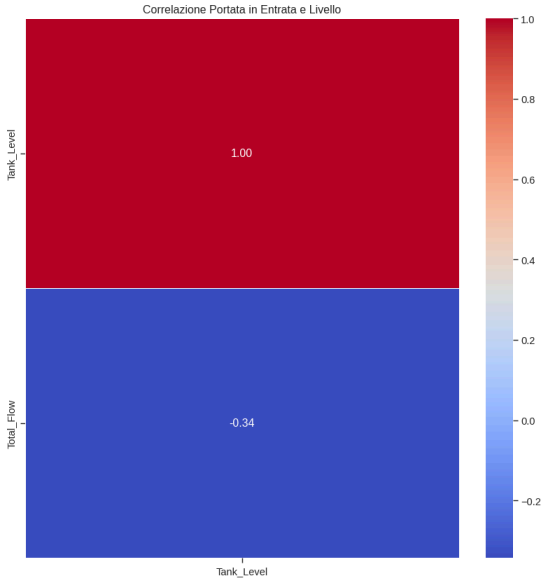


Figura 5.28: Correlazione Portata in Entrata-Livello.

```
Calulating DTW distance for Tank Level and Total Inflow
DTW Distance for Tank Level and Total Inflow (window=1000): 27.551069350517146
DTW Distance for Tank Level and Total Inflow (window=10000): 16.689119097199306
Filtering time series
DTW Distance for Tank Level and Total Inflow (filtered) (window=1000): 30.370510869755
DTW Distance for Tank Level and Total Inflow (filtered) (window=10000): 18.118642634444928
```

Figura 5.29: Distanze di DTW per Portata in Entrata-Livello (Serbatoio).

```
if os.path.exists(file_path):
    df = pd.read_csv(file_path, parse_dates=['Time'], index_col='Time')
    if os.path.exists(aggregated_file_path):
        print("Loading TOTAL_AGGREGATED_WELLS.csv")
        aggregated_df = pd.read_csv(aggregated_file_path, parse_dates=['Time'],
                index_col='Time')
        print('\nPlotting total Flows and Amps')
        plt.figure(figsize=(12, 6))
        sns.lineplot(x=aggregated_df.index, y=aggregated_df.Total_Flow,
                color='dodgerblue', label='Flows')
        sns.lineplot(x=aggregated_df.index, y=aggregated_df.Total_Amps,
                color='darkorange', label='Amps')
        plt.title('Total Flows and Amps', fontsize=14)
        plt.ylabel('Flows (l/s), Amps (A)', fontsize=14)
        plt.tight_layout()
        plt.savefig('plots/aggregated/Total_Flow_Amp.png', dpi=300)
        plt.show()
```

Dopo aver normalizzato le colonne, si calcolano correlazione (Fig. 5.28) e distanza di DTW (Fig. 5.29) tra portata totale e livello del serbatoio. I risultati sono simili a quelli con la portata in uscita, infatti la correlazione non dà informazioni di rilievo, ma le distanze di DTW (che in questo caso peggiorano dopo

il filtraggio) sono basse. I dati quindi confermano la struttura fisica del sistema, ovvero il serbatoio, che funge da punto di snodo per la rete, dipende strettamente da portata in entrata e in uscita.

```
# ---CORRELATION B/W TANK LEVEL AND TOTAL FLOW
plt.figure(figsize=(10, 10))
corr = df[['Tank_Level', 'Total_Flow']].corr()
sns.heatmap(corr[['Tank_Level']], annot=True, cmap='coolwarm', fmt='.2f',
        linewidths=0.5)
plt.title('Tank Level and Total Inflow Correlation')
plt.savefig('plots/aggregated/Correlation_Level_Inflow.png', dpi=300)
plt.show()
```

```
# ---DTW DISTANCE FOR TANK LEVEL AND TOTAL FLOW---
dtw_profile = []
window1 = 1000
window2 = 10000
Level = np.array(df['Tank_Level'].values, dtype=np.double)
Inflow = np.array(df['Total_Flow'].values, dtype=np.double)
distance1 = dtw.distance_fast(Level, Inflow, use_pruning=True, window=window1)
distance2 = dtw.distance_fast(Level, Inflow, use_pruning=True, window=window2)
dtw_profile.append({
    'Parameters': 'Level-Inflow',
    f'DTW (window: {window1})': distance1,
    f'DTW (window: {window2})': distance2
})
print(f'DTW Distance for Tank Level and Total Inflow (window={window1}): {
    distance1}')
print(f'DTW Distance for Tank Level and Total Inflow (window={window2}): {
    distance2}')
# Apply Hampel filter
print('Filtering time series')
result_Tank_Level = hampel(df['Tank_Level'], window_size=72)
filtered_Tank_Level = result_Tank_Level.filtered_data
result_Total_Flow = hampel(df['Total_Flow'], window_size=72)
filtered_Total_Flow = result_Total_Flow.filtered_data
result_Total_Amps = hampel(df['Total_Amps'], window_size=72)
filtered_Total_Amps = result_Total_Amps.filtered_data
Level = np.array(filtered_Tank_Level.values, dtype=np.double)
Inflow = np.array(filtered_Total_Flow.values, dtype=np.double)

distance1_filtered = dtw.distance_fast(Level, Inflow, use_pruning=True, window=
    window1)
distance2_filtered = dtw.distance_fast(Level, Inflow, use_pruning=True, window=
    window2)
dtw_profile.append({
    'Parameters': 'Level-Inflow (filtered)',
    f'DTW (window: {window1})': distance1_filtered,
    f'DTW (window: {window2})': distance2_filtered
})
print(f'DTW Distance for Tank Level and Total Inflow (filtered) (window={window1
    }): {distance1_filtered}')
print(f'DTW Distance for Tank Level and Total Inflow (filtered) (window={window2
    }): {distance2_filtered}')
# Keep best results
if distance2_filtered < distance2:
    df['Tank_Level'] = filtered_Tank_Level.values
    df['Total_Flow'] = filtered_Total_Flow.values
    df['Total_Amps'] = filtered_Total_Amps.values
dtw_df = pd.DataFrame(dtw_profile)
dtw_df.to_csv('plots/aggregated/dtw_inflow_profile.csv')
```

5.3 Ottimizzazione

Il sistema di ottimizzazione deve risolvere il seguente problema:

"Basandosi su una previsione della richiesta idrica e sul livello dell'acqua nel serbatoio, qual è il modo migliore di attivare le pompe dei pozzi per soddisfare la richiesta e ridurre al minimo il consumo energetico, rispettando l'ordine di priorità dei pozzi?"

L'ottimizzatore lavora in due step principali:

1. Prevedere la richiesta d'acqua futura.
2. Minimizzare la corrente totale rispettando i vincoli.

I vincoli sono:

- Stimare la portata di ogni pompa a partire da corrente e livello d'acqua del pozzo.
- Selezionare i pozzi secondo un ordine prestabilito.
- Fare sì che il livello d'acqua nel serbatoio rimanga entro i limiti prestabiliti.
- La portata in entrata totale dev'essere sufficiente per soddisfare la richiesta.

Nel test che verrà discusso sono stati inclusi i dati di soltanto quattro pozzi, in questo si modo si è potuta ridurre la potenza di calcolo necessaria e si è resa più facile la comprensione dei grafici. L'ambiente della simulazione contiene quindi dati storici dei pozzi e dati simulati adatti a rappresentare l'attività di quattro pozzi. Infatti la richiesta idrica, storicamente relativa a tutti i pozzi, è stata ridotta scalando i dati reali in modo da essere adatta a quattro pozzi.

5.3.1 Test di ottimizzazione

All'inizio dello script vengono definiti i parametri dell'ottimizzatore. Si definiscono i pozzi e il numero di pompe, la corrente massima applicabile alle pompe, la capacità del serbatoio e il limite minimo (300cm). L'obiettivo è mantenere il livello dell'acqua intorno ai 350cm e sempre dentro i limiti di massimo e di minimo. L'orizzonte di forecasting è impostato a 5000 step, dove ogni step corrisponde ad un'ora.

```
PUMPS_PER_WELL = [2, 2, 2, 3]
N_WELLS = len(PUMPS_PER_WELL)
TOTAL_PUMPS = sum(PUMPS_PER_WELL)
CURRENT_MAX = 20.0
TANK_CAPACITY = 400
TANK_MIN = 300
INITIAL_TANK = 400
TARGET_TANK = 350
horizon = 5000
```

Successivamente viene importato il dataset e vengono create variabili che contengono i dati storici presi dal dataset.

```
df = pd.read_csv("MASTER_DATASET_FINAL.csv")
# PARSE TIMESTAMPS
df["Time"] = pd.to_datetime(df["Time"])
hours = df["Time"].dt.hour.to_numpy()
days = df["Time"].dt.dayofweek.to_numpy()

# PARSE LEVELS
level_columns = sorted(
    [c for c in df.columns if c.startswith("Level_")],
    key=lambda x: int(re.findall(r"\d+", x)[0]))
historical_levels = df[level_columns].to_numpy()
well_level = historical_levels[0, :].copy()

# PARSE FLOWS
flow_columns = {}
for c in df.columns:
    if c.startswith("Flow_"):
        m = re.match(r"Flow_(W\d+)_P(\d+)", c)
        if m:
            well = m.group(1)
            flow_columns.setdefault(well, []).append(c)
for w in flow_columns:
    flow_columns[w] = sorted(flow_columns[w])
historical_flows = np.column_stack([
    df[flow_columns[w]].sum(axis=1)
    for w in sorted(flow_columns)
])

# PARSE CURRENTS
amp_columns = {}
for c in df.columns:
    if c.startswith("Amp_"):
        m = re.match(r"Amp_(W\d+)_P(\d+)", c)
        if m:
            well = m.group(1)
            amp_columns.setdefault(well, []).append(c)
for w in amp_columns:
    amp_columns[w] = sorted(amp_columns[w])
historical_currents = np.column_stack([
    df[amp_columns[w]].sum(axis=1)
    for w in sorted(amp_columns)
])

# PARSE TANK VARIABLES
historical_tank = df["Tank_Level"].to_numpy()
historical_demand = df["Tank_Outflow"].to_numpy()
demand = historical_demand.copy()
```

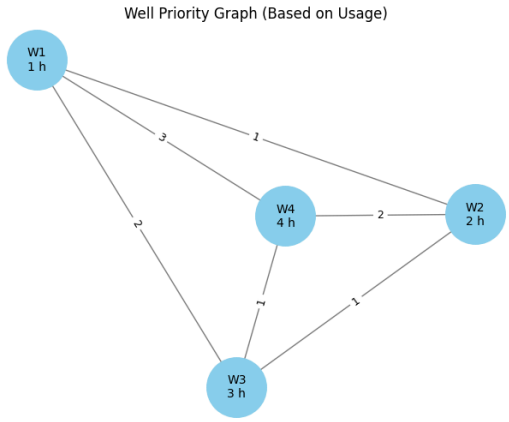


Figura 5.30: Grafo dei Pozzi.

Per rispettare l'ordine di priorità dei pozzi, essi vengono organizzati in una struttura a grafo (Fig. 5.30) dove i nodi sono i pozzi stessi e agli archi vengono associati dei costi. In questo test la priorità viene assegnata in modo casuale ma nella realtà quest'ordine andrebbe definito dal contributo totale dei pozzi: il pozzo che ha storicamente contribuito di più è il primo a essere scelto e così via fino al pozzo meno utilizzato. Il percorso del grafo con il costo minore è l'ordine ideale che l'ottimizzatore deve tentare di rispettare. Si può scegliere quanto peso dare alla priorità con il parametro *ALPHA* (valori da 0 a 1).

```
# Total operating hours for each well
usage = np.array([1, 2, 3, 4])
# Highest usage first
order = np.argsort(-usage)
priority_rank = np.empty(N_WELLS, dtype=int)
priority_rank[order] = np.arange(N_WELLS)
priority_score = N_WELLS - priority_rank

# Create directed graph
G = nx.DiGraph()
# Add nodes with usage as an attribute
for i in range(N_WELLS):
    G.add_node(
        f"W{i+1}",
        usage=usage[i]
    )
# Connect every higher-priority well to every lower-priority well
for i in range(len(order)):
    for j in range(i + 1, len(order)):
        high = order[i]
        low = order[j]
        # Weight = difference in operating hours
        weight = usage[high] - usage[low]
        G.add_edge(
            f"W{high+1}",
            f"W{low+1}",
            weight=weight
        )
ALPHA = 1    # priority importance
```

Si costruiscono diversi modelli di forecasting: uno per la previsione della richiesta e uno per ogni pozzo per la previsione dei livelli d'acqua. In questo esempio viene usato Random Forest per tutte le previsioni. Il modello per la richiesta d'acqua si addestra sui dati storici usando giorni, ore e valori traslati di 1 e di 24 step.

```
demand_df = pd.DataFrame()
demand_df["hour"] = hours
demand_df["day"] = days
demand_df["demand"] = historical_demand
demand_df["lag1"] = demand_df["demand"].shift(1)
demand_df["lag24"] = demand_df["demand"].shift(24)
demand_df.dropna(inplace=True)
X = demand_df[["hour","day","lag1","lag24"]]
y = demand_df["demand"]

demand_model = RandomForestRegressor(
    n_estimators=1,
    random_state=0
)
demand_model.fit(X,y)
```

I modelli dei pozzi si addestrano sui livelli d'acqua usando dati storici di portata e corrente, che sono i parametri a cui sono direttamente correlati.

```
rf_level = []
for w in range(N_WELLS):
    level_df = pd.DataFrame({
        "level": historical_levels[:-1,w],
        "flow": historical_flows[:-1,w],
        "current": historical_currents[:-1,w],
        "next": historical_levels[1:,w]
    })
    level_model = RandomForestRegressor(
        n_estimators=1,
        random_state=0
    )
    level_model.fit(
        level_df[["level","flow","current"]],
        level_df["next"]
    )
    rf_level.append(level_model)
```

Prima della simulazione vengono inizializzate altre variabili che conterranno i risultati delle previsioni di ogni step: al termine dell'esecuzione saranno usate per controllare i risultati.

```
tank = INITIAL_TANK
tank_history = []
forecast_history = []
pump_history = []
current_history = []
flow_history = []
current_usage_history = []
cumulative_current_usage = 0
pump_selection_history = []
well_history = []
total_cost = 0
```

Ad ogni step della simulazione si prevede la richiesta d'acqua tra un'ora e si reimpostano le pompe.

```
for t in range(len(demand_df)-horizon,len(demand_df)):
    row = X.iloc[t:t+1]
    predicted_demand = demand_model.predict(row)[0]
    forecast_history.append(predicted_demand)
    available_flow = 0
    pumps_on = np.zeros(N_WELLS)
```

L'ottimizzazione è implementata con la funzione *minimize()* della libreria *scipy*, che prende in input la funzione obiettivo da minimizzare, il metodo, i limiti e i vincoli.

La funzione obiettivo è definita nella funzione *objective()* che calcola l'energia da fornire a ciascuna pompa basandosi sull'ordine di priorità del grafo.

I limiti sono la corrente massima applicabile ad ogni pompa.

Il vincolo è definito nella funzione *flow_constraint()* che restituisce la differenza tra portata in ingresso stimata e portata richiesta. L'ottimizzatore tenta di eguagliare i valori in modo da avere differenza nulla e quindi mantenere il livello nel serbatoio stabile. La portata richiesta infatti dipende dalla portata in uscita prevista, il livello nel serbatoio e la soglia da mantenere. La portata in entrata viene stimata dalla funzione *pump_flow()*, che calcola la portata in base a corrente e livello d'acqua attualmente nel pozzo.

```
    def pump_flow(level, current):
        return 0.001 * (current**2) * level

    def optimize_currents(tank, demand, well_level, tank_min):
        n = len(well_level)
        required_flow = max(demand + TARGET_TANK - tank, 0)

        def objective(currents):
            energy = np.sum(currents**2)
            reward = np.sum(priority_score * currents)
            return energy - ALPHA * reward

        def flow_constraint(currents):
            return np.sum(pump_flow(well_level, currents)) -
                                    required_flow
        bounds = [(0, CURRENT_MAX)] * n

        x0 = np.ones(n)

        result = minimize(
            objective,
            x0,
            method="SLSQP",
            bounds=bounds,
            constraints=[
                {
                    "type": "eq",
                    "fun": flow_constraint,
                }
            ],)

        if not result.success:
            currents = np.ones(n) * CURRENT_MAX
        else:
            currents = result.x
        return currents

    currents = optimize_currents(tank, predicted_demand, well_level,
                                 TANK_MIN,)
```

Una volta definite le correnti ottimali si selezionano le pompe di ogni pozzo in modo da mantenere un utilizzo equo.

```
    selected_pumps = []
    for w in range(N_WELLS):
        if currents[w] > 0.1:
            # Choose the pump with the lowest accumulated runtime
            idx = np.argmin(pump_runtime[w])
            # Update its runtime
            pump_runtime[w][idx] += 1
        else:
            idx = None
        selected_pumps.append(idx)
        pump_selection_history.append(selected_pumps)
```

I livelli dei pozzi si aggiornano effettuando un forecast che usa le portate e le correnti appena calcolate, così da poter usare il livello previsto nel prossimo step.

```
    flows = pump_flow(well_level, currents)

    for w in range(N_WELLS):
        x = pd.DataFrame({
            "level":[well_level[w]],
            "flow":[flows[w]],
            "current":[currents[w]],
        })
        well_level[w] = rf_level[w].predict(x)[0]
    well_history.append(well_level.copy())
```

Infine si aggiorna il livello d'acqua nel serbatoio facendo la differenza tra portate in entrata e in uscita.

```
tank = tank + flows.sum() - demand[t]
tank = np.clip(tank, 0, TANK_CAPACITY)
```

5.3.2 Risultati

La valutazione dei risultati è supportata dai grafici contenenti i risultati delle previsioni e dell'ottimizzazione. Questi risultati sono relativi a un contesto in cui si hanno quattro pozzi e una domanda scalata in modo da essere soddisfacibile da questo numero di pozzi. Nei seguenti grafici saranno visualizzati i risultati di ogni step di ottimizzazione. Ogni step rappresenta un'ora e l'intera simulazione si svolge in 5000 step. I modelli di forecasting sono stati addestrati sulla porzione di dataset precedente al primo step della simulazione e testati sui dati reali indicizzati a partire dal primo timestep della simulazione.

Dalla Figura 5.31 si può osservare che il modello (Random Forest) prevede con buona precisione il fabbisogno idrico futuro, quindi l'ottimizzatore parte da una base corretta. A supporto di questa conclusione, le metriche di valutazione calcolate per Random Forest, MAE, RMSE, R2 e MAPE (Fig. 5.40) mostrano che l'errore commesso dal modello è basso sia nella previsione della domanda, che dei livelli dei pozzi. Le previsioni dei livelli prodotte dai modelli addestrati sui dati storici dei pozzi sono visualizzate in Figura 5.32, che mostra l'evolversi delle condizioni del pozzo per via dell'attività delle pompe.

In Figura 5.33 si può vedere come si evolve il livello d'acqua nel serbatoio durante i 5000 step della simulazione. Il livello rimane sempre dentro i limiti imposti e si aggira intorno al target di 350 cm, con delle variazioni dovute alle oscillazioni della domanda.

La Figura 5.34 mostra l'allocazione delle correnti ottime durante la simulazione e si nota che l'ordine di priorità viene rispettato: tutti i pozzi vengono attivati ma viene data più corrente ai pozzi di priorità più alta (in questo caso l'ordine è 4, 3, 2, 1). I picchi di corrente si trovano in corrispondenza dei picchi nella richiesta idrica. Si può vedere che queste correnti sono il minimo necessario per soddisfare la richiesta nella Figura 5.35, che mostra il contributo di ogni pozzo alla portata complessiva, mettendola a confronto con la domanda totale (linea nera tratteggiata), che è sempre soddisfatta. Inoltre, le pompe sono usate in modo equo come si può osservare dalla Figura 5.36.

In Figura 5.37 si mette a confronto il consumo di corrente ottenuto dalla gestione manuale delle pompe con quello ottenuto dopo l'ottimizzazione. La corrente totale consumata dalla gestione manuale è di circa 116.446 A (calcolata sommando cumulativamente le correnti storiche di tutte le pompe dei quattro pozzi), mentre la corrente totale ottimizzata è circa 82.931 A, producendo un risparmio energetico almeno del 30%.

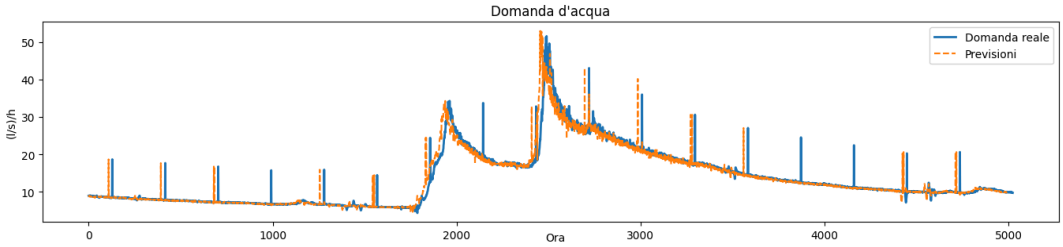


Figura 5.31: Previsione richiesta idrica.

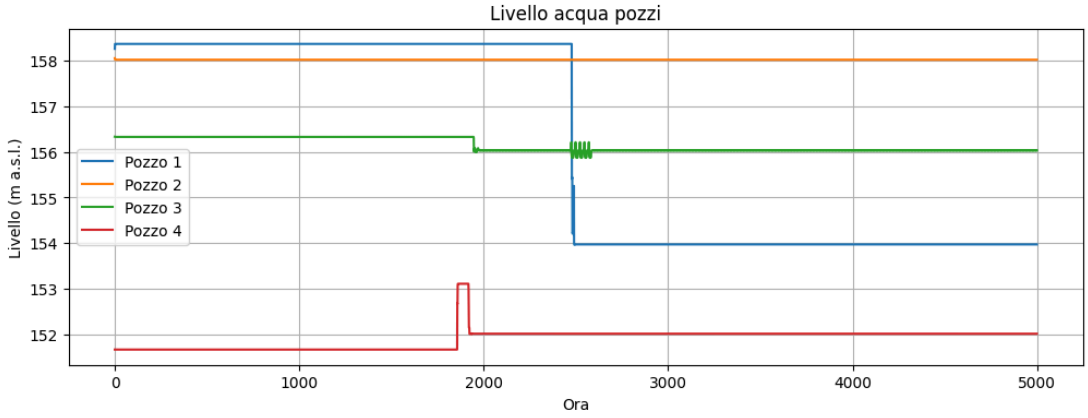


Figura 5.32: Previsione Livelli nei Pozzi.

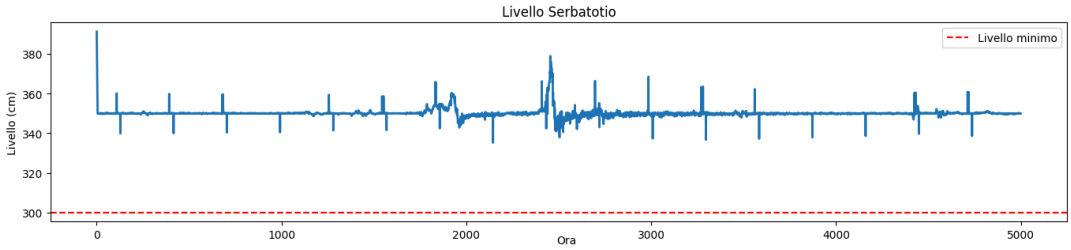


Figura 5.33: Andamento del Livello del Serbatoio durante l'ottimizzazione.

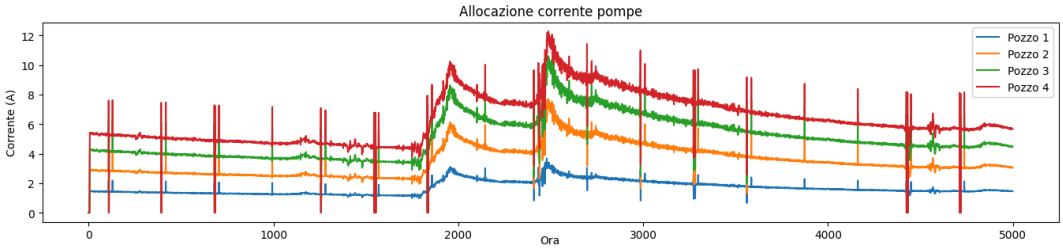


Figura 5.34: Correnti ottime allocate ad ogni pozzo.

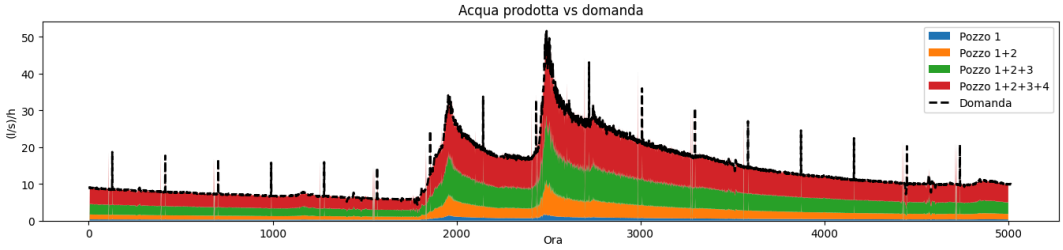


Figura 5.35: Portata in Ingresso vs. Domanda.

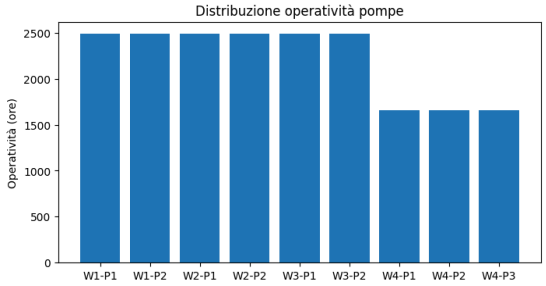


Figura 5.36: Tempo di utilizzo Pompe.

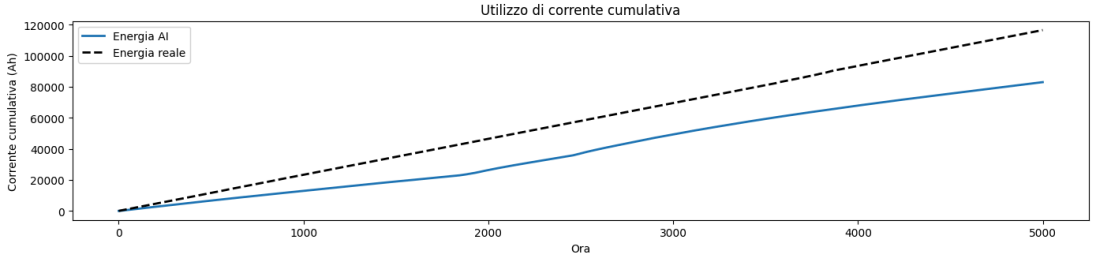


Figura 5.37: Corrente consumata manualmente vs. ottimizzata.

5.4 Benchmark

Per avere informazioni più accurate sulle prestazioni di un modello, lo si mette a confronto con altri modelli e si calcolano le metriche di valutazione. Il benchmark svolto misura le prestazioni di 3 modelli: Random Forest, XGBoost e LSTM. Le metriche utilizzate sono: MAE, RMSE, R2 e MAPE. Le funzioni per calcolarle sono state importate da *sklearn*.

```
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score
)
def calculate_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    mask = y_true != 0
    if np.any(mask):
        mape = np.mean(
            np.abs((y_true[mask] - y_pred[mask])/ y_true[mask])
        ) * 100
    else:
        mape = np.nan
    return {
        "MAE": mae,
        "RMSE": rmse,
        "R2": r2,
        "MAPE (%)": mape
    }
```

Per usare LSTM è necessario normalizzare i dati e creare sequenze dalle time series.

```
def scale_data(df):
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(df)
    return scaled_data, scaler

def create_sequences(data, window):
    X, y = [], []
    for i in range(len(data) - window):
        X.append(data[i:i + window])
        y.append(data[i + window, 0])
    return np.array(X), np.array(y)
```

I grafici (Fig. 5.38, 5.39) rappresentano visivamente i risultati delle previsioni di ogni modello confrontati con i valori reali della domanda e dei livelli. In una prima analisi dei risultati si può dire che Random Forest segue in modo piuttosto accurato i valori reali nella maggior parte dei casi, commettendo più errori nella previsione dei livelli del pozzo 3. Le previsioni di XGBoost combaciano con i valori effettivi della domanda, ma non con quelli dei pozzi. LSTM invece, funziona bene nella previsione dei livelli, ma non è stato in grado di modellare la domanda.

Le metriche di valutazione forniscono un riscontro quantitativo delle osservazioni precedenti. Il riepilogo riportato in Figura 5.40 mostra le prestazioni dei tre modelli nella previsione delle cinque variabili considerate. Nel complesso, Random Forest presenta i valori più bassi di MAE, RMSE e MAPE nella maggior parte dei casi e valori di R^2 positivi per tutte le variabili analizzate. XGBoost mostra invece prestazioni inferiori nella previsione dei livelli dei pozzi, come evidenziato dai valori negativi di R^2 ottenuti per i pozzi 1, 2 e 3. LSTM fornisce buoni risultati nella previsione dei livelli dei pozzi 2, 3 e 4, ma presenta prestazioni insoddisfacenti nella previsione della domanda idrica e del livello del pozzo 1, per i quali il valore di R^2 risulta negativo. Considerando congiuntamente le diverse metriche e tutte le variabili analizzate, Random Forest risulta quindi il modello più affidabile e performante tra quelli testati.

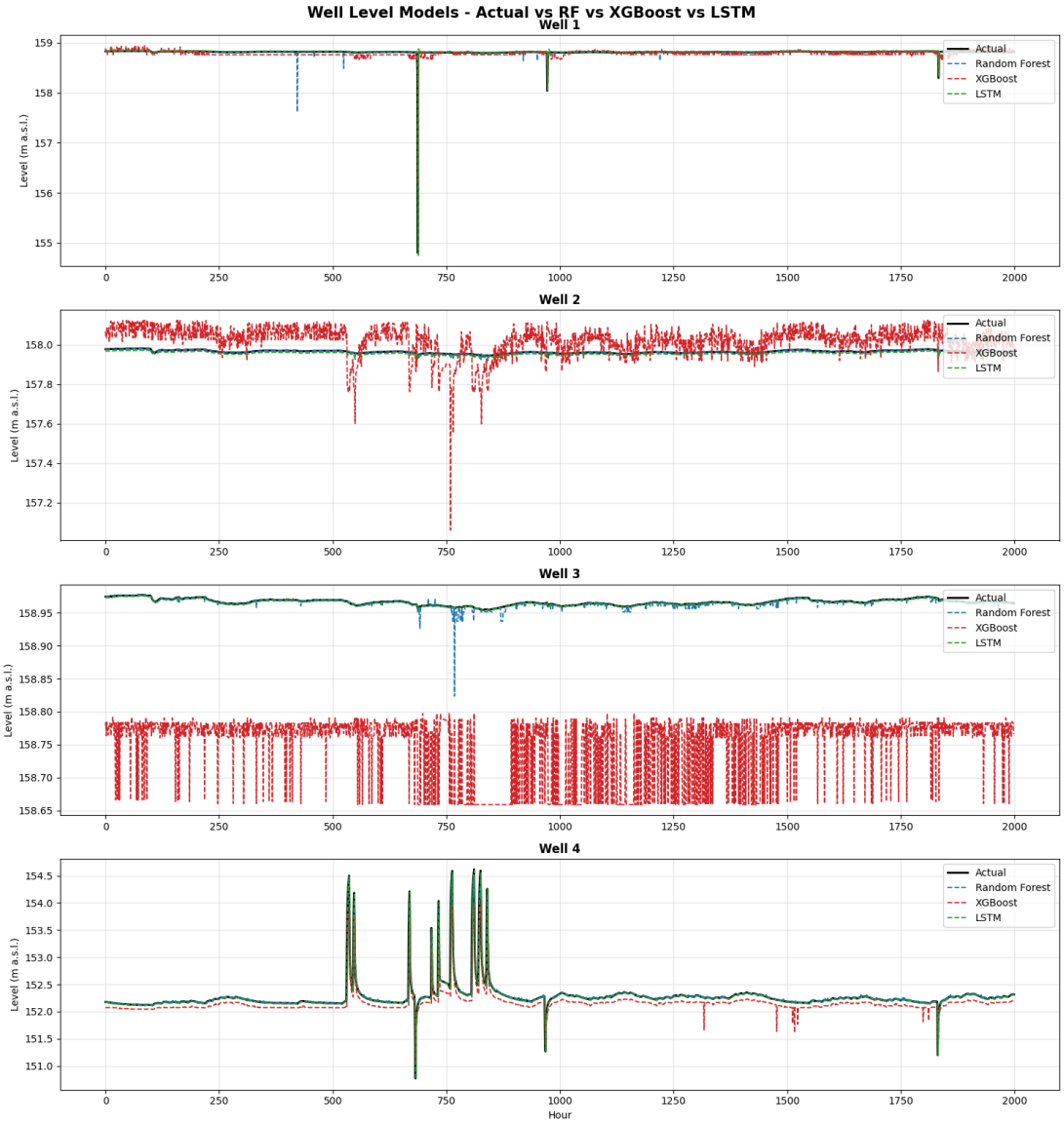


Figura 5.38: Benchmark Pozzi.

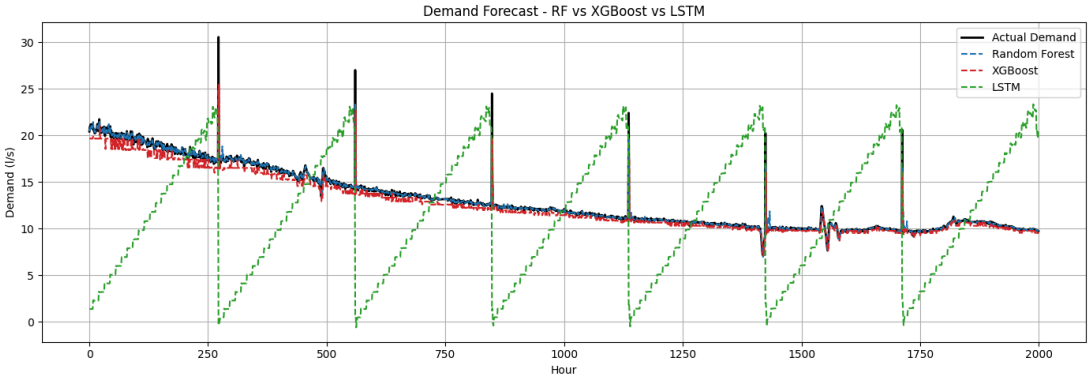


Figura 5.39: Benchmark Domanda.

```
============================================================
MODEL PERFORMANCE SUMMARY
============================================================

Demand Models:
        Model    MAE   RMSE      R2  MAPE (%)
Random Forest 0.0846 0.2751  0.9930    0.6087
      XGBoost 0.5210 0.9737  0.9128    3.6594
         LSTM 6.6791 7.9902 -4.8690   52.5372

Well Level Models:
  Well         Model    MAE   RMSE         R2  MAPE (%)
Well 1 Random Forest 0.0044 0.0831     0.2038    0.0028
Well 1       XGBoost 0.0423 0.1136    -0.4892    0.0267
Well 1          LSTM 0.0196 0.1259    -0.8293    0.0124
Well 2 Random Forest 0.0004 0.0025     0.8883    0.0003
Well 2       XGBoost 0.0767 0.0887  -135.3945    0.0486
Well 2          LSTM 0.0092 0.0093    -0.5044    0.0059
Well 3 Random Forest 0.0010 0.0042     0.1804    0.0006
Well 3       XGBoost 0.2184 0.2236 -2323.6772    0.1374
Well 3          LSTM 0.0012 0.0013     0.9201    0.0008
Well 4 Random Forest 0.0259 0.1181     0.8156    0.0170
Well 4       XGBoost 0.1203 0.1694     0.6206    0.0789
Well 4          LSTM 0.0281 0.1249     0.7938    0.0184
```

Figura 5.40: Metriche di Valutazione.

Capitolo 6

Discussioni e Conclusioni

Nel presente progetto di tesi si è proposta una metodologia di ottimizzazione dei pozzi della sorgente Gorgovivo. Dallo studio delle variabili del sistema si è costruito un sistema di ottimizzazione che sfrutta modelli di intelligenza artificiale per prevedere il fabbisogno idrico e i livelli d'acqua nei pozzi e attiva le pompe dei pozzi in modo da ridurre al minimo il consumo di energia elettrica.

A concludere questo lavoro si evidenzia come l'integrazione del machine learning in sistemi reali possa avere un impatto positivo sulla loro efficienza e sostenibilità. Durante il percorso è stata posta particolare attenzione all'analisi dei dati che, se svolta correttamente, permette di ricavare nuove conoscenze sul dominio in esame, che vengono poi applicate per sviluppare il nucleo del progetto. Una difficoltà riscontrata durante l'analisi è stata ottenere dati di qualità, che tutt'ora risultano essere carenti dal punto di vista della completezza, infatti le analisi e i test svolti partono da un dataset limitato, che ha richiesto l'utilizzo di interpolazione per approssimare i valori reali. Inoltre, nei test sono stati inclusi i dati di soltanto quattro pozzi anziché tredici. Questo ha consentito di ridurre la complessità dei test e l'utilizzo di modelli di IA computazionalmente leggeri che non richiedessero di essere addestrati con la potenza di calcolo di una GPU, ma d'altra parte così facendo si ottiene una modellazione semplificata rispetto alla realtà. Sempre per questo motivo si sono dovuti simulare alcuni dati per rispecchiare una situazione in cui si hanno soltanto quattro pozzi attivi: i dati storici della richiesta idrica fanno riferimento al sistema completo, ma per svolgere la simulazione con quattro pozzi è stata scalata in modo da essere adeguata al caso.
Il sistema fisico è influenzato da anche altri fattori dei quali è stata considerata soltanto una parte. Ad esempio, tutti i pozzi sono posti a distanze diverse rispetto il serbatoio, quindi si dovrebbe considerare un ritardo tra l'attivazione delle pompe e il momento in cui l'acqua raggiunge il serbatoio.
Ogni step di forecasting equivale a un'ora, ma durante l'operatività del sistema può essere preferibile usare orizzonti di lunghezze dinamiche, sia perché la richiesta idrica può variare più o meno frequentemente a seconda del periodo, sia per

dare alle pompe un periodo di operatività più o meno lungo prima di alternarle, così da facilitarne la manutenzione.
La scelta del modello di forecasting è cruciale per il rispetto dell'obiettivo del progetto, ovvero il risparmio energetico. In questo studio preliminare sono stati usati perlopiù modelli di machine learning, ma potrà essere esteso aggiungendo le variabili di tutti i pozzi e si avrà la necessità di modelli più potenti basati su deep learning che possano dare risultati di efficientamento ottimi anche all'aumentare della complessità del problema.
Anche considerando queste limitazioni, il modello sviluppato soddisfa l'obiettivo principale di risparmio energetico che si era posto e sfrutta i dati storici per simulare l'attivazione dei pozzi e, in questi test, ha permesso di ottenere fino al 30% di risparmio di energia elettrica rispetto alla gestione manuale, che è una percentuale promettente per il futuro del progetto ma è sicuramente dipendente dalle condizioni specifiche in cui si sono svolti i test. Nonostante siano necessarie diverse modifiche, sia ai dati in input, che all'implementazione del modello di forecasting, questo sistema di ottimizzazione potrà essere ampliato in modo da includere tutti i pozzi della sorgente per simulare l'operatività dell'intero acquedotto e potrà fornire risultati più specifici.

Lavorare a questo progetto è stata una grande opportunità per studiare e conoscere metodi di analisi e di forecasting all'avanguardia, che possono essere applicati in numerosi casi d'uso e che se applicati correttamente si rivelano essere estremamente utili in un periodo in cui la sostenibilità ecologica e la salvaguardia delle risorse idriche sono diventate indispensabili.

6.1 Sviluppi futuri

Questo lavoro è parte di un progetto sviluppato in collaborazione con VIVA Servizi S.p.A. e costituisce una base che può essere integrata a un sistema reale. I risultati dei test dimostrano che un approccio data-driven basato su machine learning risulta essere efficace se fondato su un'analisi esaustiva dei dati ma, per rendere il sistema più dinamico e per migliorare le previsioni, si possono sfruttare i dati a disposizione in modo ancora più approfondito.

L'attività dell'acquedotto è definita da comportamenti umani abitudinari, che si riflettono nella richiesta d'acqua durante la giornata e durante l'anno. L'inclusione di informazioni aggiuntive di questo tipo nel modello di ottimizzazione può migliorare le prestazioni in situazioni particolari. Ad esempio si può voler aumentare la distribuzione d'acqua in periodi di maggiore utilizzo, come d'estate, e magari ridurla durante le ore della giornata in cui si osserva meno richiesta.

Un altro modo in cui si può aggiornare il modello è con l'inclusione di diversi profili di ottimizzazione. Al momento il modello cerca sempre di mantenere il livello d'acqua nel serbatoio entro dei limiti sicuri e moderatamente ampi, ma a seconda della situazione si potrebbe voler aumentare l'aggressività della distribuzione e consentire uno svuotamento del serbatoio più ampio, oppure in periodi che lo consentono, di accumulare più acqua possibile nel serbatoio in vista di siccità nel futuro prossimo.

Una volta ampliate le simulazioni in modo da comprendere tutti i pozzi, messi alla prova ulteriori modelli di forecasting (in particolare di deep learning) sul dataset completo e testato il modello di ottimizzazione in diverse condizioni, se i risultati lo consentono si potrà passare da un ambiente digitale a uno fisico integrando il modello ai sistemi reali dell'acquedotto.

Bibliografia

[1] U.S. Geological Survey Water Science School. How much water is there on earth?, 2019. https://www.usgs.gov/water-science-school/science/how-much-water-there-earth (visitato il giorno 11/10/2026).

[2] Eurostat. Energy statistics - an overview, 2025. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Energy_statistics_-_an_overview (visitato il giorno 11/10/2026).

[3] Time series decomposition methods. https://www.kaggle.com/code/alisadeghiaghili/time-series-decomposition-methods (visitato il giorno 11/10/2026).

[4] Hicham Janati and Johann Faouzi. Sakoe–chiba band. https://pyts.readthedocs.io/en/latest/auto_examples/metrics/plot_sakoe_chiba.html (visitato il giorno 11/10/2026).

[5] Hampel filter. https://it.mathworks.com/help/dsp/ref/hampelfilter.html (visitato il giorno 11/10/2026).

[6] Fatemeh Ghobadi and Doosun Kang. Application of machine learning in water resources management: A systematic literature review. *Water*, 15(4), 2023. 10.3390/w15040620.

[7] Hilla Shemer, Shlomo Wald, and Raphael Semiat. Challenges and solutions for global water scarcity. *Membranes*, 13(6), 2023. 10.3390/membranes13060612.

[8] Yuyao Chen, Wei Gong, Christian Obrecht, and Frédéric Kuznik. A review of machine learning techniques for building electrical energy consumption prediction. *Energy and AI*, 21:100518, 2025. https://doi.org/10.1016/j.egyai.2025.100518.

[9] Ashraf A. Ahmed, Sakina Sayed, Antoifi Abdoulhalik, Salissou Moutari, and Lukumon Oyedele. Applications of machine learning to water resources management: A review of present status and future

opportunities. *Journal of Cleaner Production*, 441:140715, 2024. https://doi.org/10.1016/j.jclepro.2024.140715.

[10] Consorzio Gorgovivo. https://gorgovivo.it/ (visitato il giorno 11/10/2026).

[11] Alessandro Galdelli, Gagan Narang, Lucia Migliorelli, Antonio Domenico Izzo, Adriano Mancini, and Primo Zingaretti. An ai-driven prototype for groundwater level prediction: Exploring the gorgovivo spring case study. In Gian Luca Foresti, Andrea Fusiello, and Edwin Hancock, editors, *Image Analysis and Processing – ICIAP 2023*, pages 418–429, Cham, 2023. Springer Nature Switzerland. 10.1007/978-3-031-43153-1_35.

[12] Amal Tawakuli, Bastian Havers, Vincenzo Gulisano, Daniel Kaiser, and Thomas Engel. Survey:time-series data preprocessing: A survey and an empirical analysis. *Journal of Engineering Research*, 13(2):674–711, 2025. https://doi.org/10.1016/j.jer.2024.02.018.

[13] Che cosa sono i modelli arima? https://www.ibm.com/it-it/think/topics/arima-model (visitato il giorno 11/10/2026).

[14] Phd Dr Aakash Ramchand Dil. Testing for time series stationarity: A practical guide to the dickey-fuller and augmented dickey-fuller tests. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5287311 (visitato il giorno 11/10/2026).

[15] Patrick Schober, Christa Boer, and Lothar A. Schwarte. Correlation coefficients: Appropriate use and interpretation. *Anesthesia & Analgesia*, 126(5):1763–1768, May 2018. 10.1213/ane.0000000000002864.

[16] Janse RJ, Hoekstra T, Jager KJ, Zoccali C, Tripepi G, Dekker FW, and van Diepen M. Conducting correlation analysis: important limitations and pitfalls, 2021. 10.1093/ckj/sfab085.

[17] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978. 10.1109/TASSP.1978.1163055.

[18] Toshiko Kaneda and Jason Bremner. Understanding population projections: Assumptions behind the numbers, 2014. https://assets.prb.org/pdf14/understanding-population-projections.pdf (visitato il giorno 11/10/2026).

[19] Wenxiang Li and K. L. Eddie Law. Deep learning models for time series forecasting: A review. *IEEE Access*, 12:92306–92327, 2024. 10.1109/ACCESS.2024.3422528.

[20] Dariush Pourkeramati, Gareth Wadge, Rachel Hassall, Charlotte Mitchell, Anish Khadka, Shiwang Jaiswal, Andrew Duncan, and Rossella Arcucci. Emerging paradigms in the energy sector: Forecasting and system control optimisation. *Progress in Engineering Science*, 3(2):100260, 2026. https://doi.org/10.1016/j.pes.2026.100260.

[21] Cos'è la previsione finanziaria? https://www.ibm.com/think/topics/financial-forecasting (visitato il giorno 11/10/2026).

[22] J. Scott Armstrong and Julia A. Trevarthen. Forecasting for environmental decision making, 1999. 10.1007/978-1-4612-1418-2_7.

[23] Julie C. Lauffenburger, Jessica M. Franklin, Alexis A. Krumme, William H. Shrank, Olga S. Matlin, Claire M. Spettell, Gregory Brill, and Niteesh K. Choudhry. Predicting adherence to chronic disease medications in patients with long-term initial medication fills using indicators of clinical events and health behaviors. *Journal of Managed Care & amp; Specialty Pharmacy*, 24(5):469–477, May 2018. 10.18553/jmcp.2018.24.5.469.

[24] Kasun Mendis, Manjusri Wickramasinghe, and Pasindu Marasinghe. Multivariate time series forecasting: A review. In *Proceedings of the 2024 2nd Asia Conference on Computer Vision, Image Processing and Pattern Recognition*, CVIPPR 2024, page 1–9. ACM, April 2024. 10.1145/3663976.3664241.

[25] Qianqian Yu, Guang Yang, Xiao Wang, Yaxin Shi, Ying Feng, and Ang Liu. A review of time series forecasting and spatio-temporal series forecasting in deep learning, 2025. https://doi.org/10.1007/s11227-025-07632-w.

[26] What is long short-term memory (lstm)? https://www.ibm.com/think/topics/lstm (visitato il giorno 11/10/2026).

[27] Ariele Zanfei, Bruno M. Brentan, Andrea Menapace, Maurizio Righetti, and Manuel Herrera. Graph convolutional recurrent neural networks for water demand forecasting. *Water Resources Research*, 58(7), July 2022. 10.1029/2022wr032299.

[28] Evangelos Rozos. Machine learning, urban water resources management and operating policy. *Resources*, 8(4), 2019. 10.3390/resources8040173.