ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Розробка інформаційної системи прогнозування споживання електроенергії»

(тема кваліфікаційної роботи українською мовою)

«Development of an Information System for Electricity Consumption Forecasting»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 122 Комп'ютерні науки
(код, назва спеціальності)
Освітня програма Комп'ютерні науки
(назва)
САВЕНКО Микита Сергійович
(прізвище, ім'я, по-батькові здобувача)

Керівник к.т.н., доцент Фразе-Фразенко О.О. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент к.т.н., начальник ІОЦ ОНЕУ, Домаскін О.М.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій
№ ___ від ___ _____ 2025 р.

Завідувачка кафедри

_____ КАЗАКОВА Надія
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № ___ від _____ 2025 р.

Оцінка _____ / _____ / _____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ КОПИЧЕНКО Іван
(підпис) (прізвище,ім'я)

Одеса 2025

АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі розглянуто процес розробки програмної системи прогнозування споживання електроенергії з використанням моделей часових рядів. Актуальність теми зумовлена необхідністю підвищення ефективності управління енергетичними ресурсами в умовах зростання попиту та нестабільності навантажень.

У роботі виконано аналіз математичних моделей прогнозування, зокрема ARIMA, SARIMA та Prophet, з акцентом на їх застосування для обробки часових рядів, що мають сезонні та трендові компоненти. Запропонована архітектура системи охоплює модулі для обробки даних, побудови моделі та візуалізації результатів. Програмна реалізація виконана мовою Python із використанням бібліотек Pandas, StatsModels, Prophet, а також matplotlib і Seaborn для графічного представлення даних.

Система працює у консольному середовищі та забезпечує повний цикл: від завантаження даних до отримання прогнозу та оцінки його точності. Для перевірки ефективності реалізовано порівняльне тестування SARIMA та Prophet на одному датасеті. За результатами моделювання визначено, що обидві моделі забезпечують задовільну точність, однак Prophet демонструє кращу адаптацію до трендів без попереднього перетворення ряду.

Результати роботи можуть бути використані для створення аналітичних систем прогнозування енергоспоживання, автоматизованого керування навантаженнями, а також як основа для подальших досліджень у сфері прогнозування часових рядів.

ABSTRACT

This bachelor's thesis presents the development of a software system for forecasting electricity consumption using time series models. The relevance of the topic is driven by the growing need for efficient energy management under conditions of increasing demand and unstable load profiles.

The work includes an in-depth analysis of forecasting models such as ARIMA, SARIMA, and Prophet, focusing on their suitability for time series with seasonal and trend components. The proposed system architecture consists of data processing, modeling, and visualization modules. The implementation was carried out in Python using libraries such as Pandas, StatsModels, Prophet, matplotlib, and Seaborn.

The application operates in a console-based environment and performs a full prediction cycle: from loading data to generating a forecast and evaluating its accuracy. Comparative testing of SARIMA and Prophet was performed on the same dataset. Results demonstrate that both models achieve acceptable accuracy, with Prophet showing better adaptability to trend components without the need for data transformation.

The developed system can be used as a foundation for analytical platforms for energy consumption forecasting, automated load control, and further research in the field of time series forecasting.

ЗМІСТ

ВСТУП 6
1 ПІДХОДИ ДО ПРОГНОЗУВАННЯ НА ОСНОВІ ЧАСОВИХ РЯДІВ 8
1.1. Поняття та властивості часових рядів 8
1.2. Класифікація та компоненти часових рядів 11
1.3. Проблема стаціонарності та способи її подолання 14
1.4. Основні моделі прогнозування часових рядів 18
1.5. Застосування моделей для прогнозування енергоспоживання 21
2. ОБГРУНТУВАННЯ ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ 25
2.1. Вимоги до інструментального середовища 25
2.2. Вибір мови програмування та середовища розробки 26
2.3. Бібліотеки для обробки даних і моделювання часових рядів 28
2.4. Інструменти для візуалізації результатів 31
2.5. Засоби збору та зберігання даних 33
3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ЗАСТОСУНКУ 35
3.1. Вимоги до моделі прогнозування 35
3.2 Архітектура системи прогнозування 36
3.3 Алгоритм обробки часових рядів 42
4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ 44
4.1. Структура програмної реалізації 44
4.2. Завантаження і попередня обробка даних 49
4.3. Реалізація моделей прогнозування 51
4.4. Вивід і оцінка результатів 56
ВИСНОВКИ 59
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 61

ВСТУП

В умовах глобального потепління, зростання енергетичних витрат та потреби у сталому розвитку, особливої ваги набуває питання ефективного управління енергоспоживанням. Одним із напрямів такої ефективності є точне прогнозування споживання електроенергії, що дозволяє не лише знижувати фінансові витрати, а й мінімізувати негативний вплив на довкілля. Завдяки обґрунтованому прогнозуванню енергетичних потреб можливо зменшити резерви генерації, скоротити пікові навантаження на електромережу та знизити обсяги викидів CO_2 в атмосферу.

Інструменти прогнозування, побудовані на основі моделей часових рядів, зокрема ARIMA, SARIMA, методів експоненціального згладжування та сучасних підходів машинного навчання, демонструють високу ефективність у задачах передбачення на основі історичних даних. Актуальність теми також зумовлена активним розвитком смарт-енергетики, де моделі прогнозування відіграють ключову роль у побудові енергетичних стратегій, оптимізації розподілу ресурсів та підтримці енергетичної стабільності в реальному часі.

Метою дипломної роботи є розробка програмної системи для прогнозування споживання електроенергії на основі аналізу часових рядів з урахуванням сезонності, тренду та стохастичних впливів.

Для досягнення цієї мети поставлено такі завдання:

- Провести аналіз теоретичних основ та методів прогнозування часових рядів;
- Дослідити особливості застосування моделей ARIMA, SARIMA, Prophet;
- Обґрунтувати вибір інструментів для реалізації моделі;
- Розробити та реалізувати програмне забезпечення для прогнозування споживання електроенергії;
- Провести тестування моделі та оцінити її точність;

- Проаналізувати вплив використання прогнозних моделей на покращення екологічної ситуації через зменшення нераціонального використання електроенергії.

Об’єктом дослідження є процес споживання електроенергії в побутових та виробничих умовах.

Предметом дослідження є методи прогнозування значень часових рядів та програмні засоби для їх реалізації.

Методи дослідження

У дипломній роботі використано математичне моделювання, методи статистичного аналізу, машинного навчання, а також інструменти програмної реалізації на мові Python із застосуванням бібліотек Pandas, NumPy, Statsmodels, Prophet, Matplotlib та ін.

Робота містить 62 сторінок, 12 рисунків, 14 таблиць, 15 джерел посилань

1 ПІДХОДИ ДО ПРОГНОЗУВАННЯ НА ОСНОВІ ЧАСОВИХ РЯДІВ

1.1. Поняття та властивості часових рядів

У сучасних умовах цифрової трансформації великі обсяги інформації накопичуються у вигляді послідовностей спостережень, впорядкованих у часі. Такі послідовності називають часовими рядами. Вони мають надзвичайно широке застосування у прогнозуванні: від метеорології та фінансів — до енергетики та біомедичних досліджень.

Часовий ряд — це сукупність спостережень, отриманих у послідовні моменти часу. Його можна описати як функцію y_t, яка залежить від часу t:

$$y_t = \xi(t), \quad t = 1, 2, ..., n$$

де: y_t — значення змінної у момент часу t, n — кількість спостережень.

На відміну від випадкових вибірок, елементи часового ряду зазвичай мають внутрішню часову залежність. Саме ця властивість робить аналіз часових рядів специфічним — класичні методи статистики (які передбачають незалежність спостережень) часто не дають коректних результатів [1].

Часові ряди можна класифікувати за різними ознаками:

- Дискретні або неперервні (залежно від кроку часу);
- Стаціонарні або нестаціонарні (в залежності від сталість середнього, дисперсії та автокореляції);
- Сезонні, циклічні, стаціонарні з трендом тощо.

До найважливіших властивостей часових рядів належать:

Автокореляція

Це здатність даних бути пов’язаними між собою на певному часовому відрізку. Якщо значення в момент часу t пов’язане із значенням у момент t−k, говорять про автокореляцію на лагу k. Вона обчислюється за формулою:

$$\rho_k = \frac{\sum_{t=k+1}^{n}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{n}(y_t - \bar{y})^2}$$

де $\bar{y}$ — середнє значення часового ряду.

Нестаціонарність

Нестаціонарний часовий ряд — це ряд, у якого середнє значення, дисперсія або автокореляція змінюються з часом (рис. 1.1). Такий ряд не може бути напряму використаний у моделях типу ARIMA, і його необхідно стабілізувати [2].

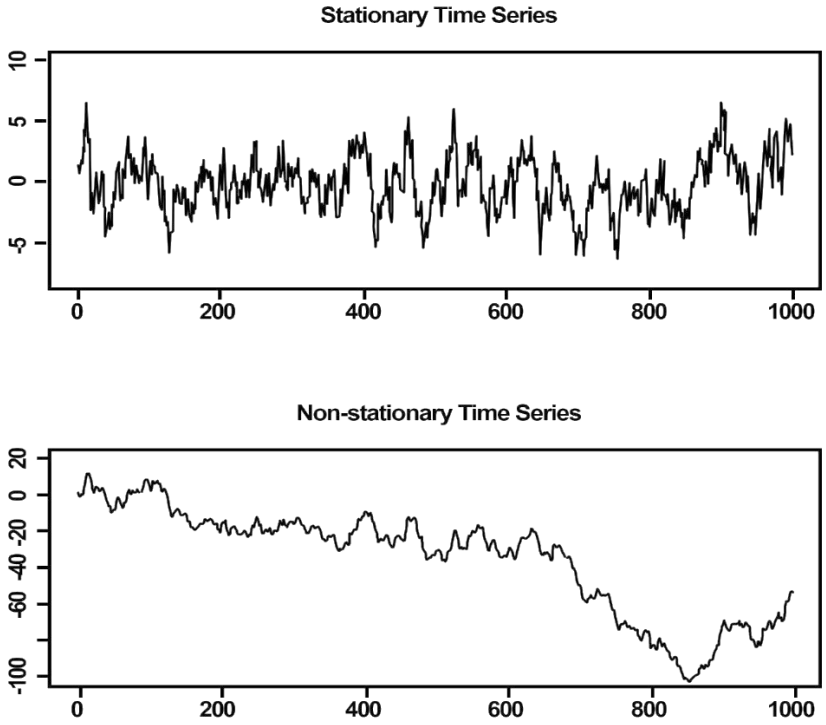

Рисунок 1.1 — Стаціонарний та нестаціонарний часові ряди

Сезонність

Це повторювані шаблони у ряді, що виникають із фіксованою частотою — наприклад, споживання електроенергії зростає взимку та зменшується влітку, або пікові навантаження припадають на вечірній час доби. Це характерна риса енергоспоживання (рисю 1.2).

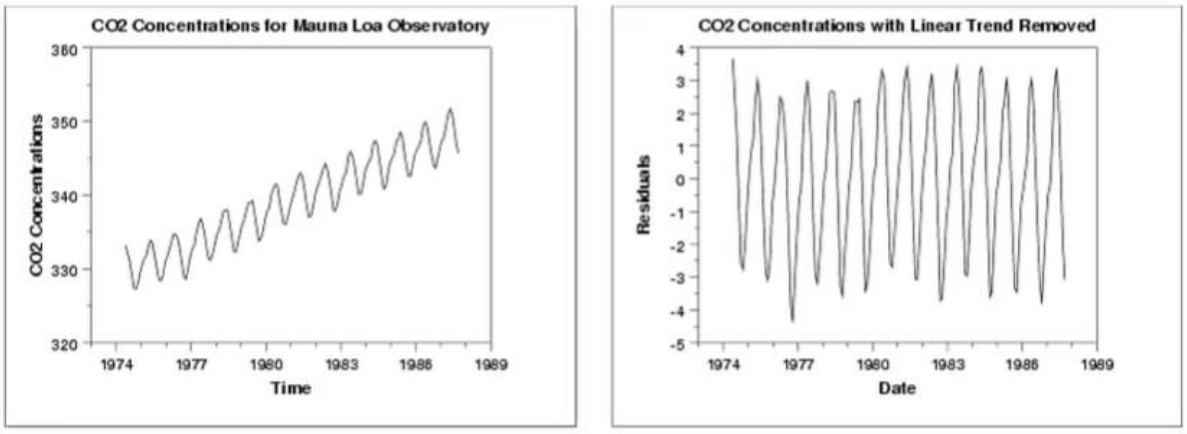

Рисунок 1.2 — Повний часовий ряд та ряд без лінійного тренду

Структурні зрушення

Це раптові зміни у рівні або поведінці часового ряду, пов’язані з політичними, соціальними, кліматичними чи техногенними факторами. Наприклад, під час пандемії COVID-19 у 2020 р. структура енергоспоживання кардинально змінилась.

Рисунок 1.3 відображає загальний вигляд ряду як суми або добутку складових: трендової, сезонної та випадкової. Вона є базовою при побудові прогнозних моделей, таких як ARIMA, SARIMA, Prophet [6].

У контексті **енергетики**, часові ряди використовуються для аналізу **добового, тижневого, сезонного та річного споживання електроенергії**. Системи обліку формують ці ряди на основі показів приладів щогодини або щохвилини. Прогнозування таких рядів дозволяє:

- Зменшити пікові навантаження,
- Планувати генерацію відповідно до попиту,
- Оптимізувати споживання за тарифними зонами,
- Мінімізувати перевитрати електроенергії й викиди CO_2 [15].

Водночас на ці ряди можуть впливати **зовнішні фактори** — погода, температура, тривалість світлового дня, а також поведінкові особливості

споживачів. Саме тому моделювання часового ряду в енергетиці потребує врахування не лише історичних даних, але й контекстуальних факторів.

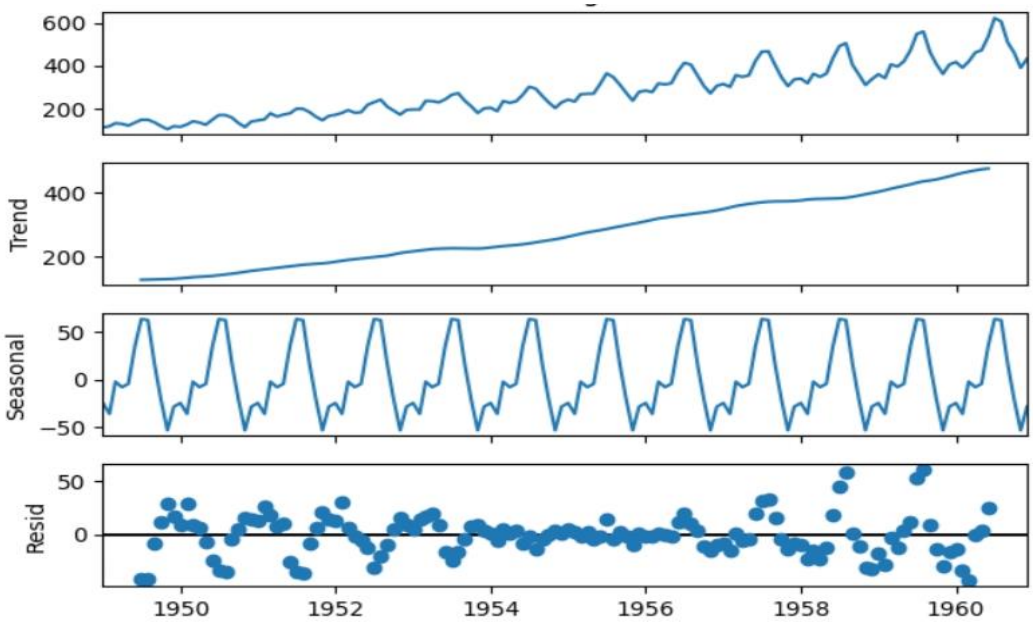

Рисунок 1.3 — Компонентна структура часового ряду.

1.2. Класифікація та компоненти часових рядів

Часові ряди в основі своїй є комплексним об’єктом, у якому поєднуються як передбачувані, так і випадкові коливання. Для точного аналізу й побудови адекватних моделей прогнозування необхідно розділити ряд на окремі компоненти, кожна з яких відображає певний тип змінності. Такий підхід дозволяє не лише покращити точність прогнозу, але й надати інтерпретацію результатів, зокрема в енергетичному або екологічному контексті.

Класифікація часових рядів може здійснюватися за різними критеріями [1]:

1. За типом спостережень:

- Дискретні — значення спостерігаються у фіксовані моменти часу (щогодини, щодня);
- Неперервні — значення фіксуються постійно, з дуже малим кроком часу (рідко зустрічаються у прикладному прогнозуванні).

2. За характером залежності:

- Стохастичні ряди — містять випадкову компоненту;
- Детерміністичні ряди — побудовані на основі чітких функціональних залежностей.

3. За структурою змінності:

- Стаціонарні — із незмінними середнім значенням, дисперсією та автокореляційною функцією;
- Нестаціонарні — зі змінними характеристиками у часі [2].

Загальноприйнята модель представлення часового ряду передбачає його розкладання на окремі компоненти. У базовому вигляді вона виглядає так:

Адитивна модель:

$$y_t = T_t + S_t + C_t + \varepsilon_t$$

Мультиплікативна модель:

$$y_t = T_t \cdot S_t \cdot C_t \cdot \varepsilon_t$$

де:

- T_t — **тренд** (довгострокова зміна рівня ряду),
- S_t — **сезонність** (регулярні повторювані коливання),
- C_t — **циклічна компонента** (довготривалі коливання без чіткої періодичності),
- ε_t — **випадкова (шумова)** компонента [1].

Адитивна модель підходить, коли коливання мають сталу амплітуду, тоді як мультиплікативна — коли амплітуда зростає з часом (наприклад, унаслідок інфляції, демографічних змін тощо) [2].

Трендова компонента

Тренд відображає довгострокову тенденцію зміни значення змінної. Він може бути:

- Зростаючим (наприклад, через загальне зростання споживання електроенергії внаслідок урбанізації);
- Спадним (наприклад, через впровадження енергозберігаючих технологій);
- Лінійним, експоненціальним або поліноміальним.

Виявлення тренду зазвичай здійснюється методами скользького середнього або регресії.

Сезонна компонента

Сезонність описує циклічні коливання з чіткою періодичністю, наприклад:

- добовий цикл (споживання вище вдень, нижче — вночі),
- тижневий (вихідні/робочі дні),
- річний (опалювальний сезон) [6].

Опис сезонності є критично важливим для прогнозування у сфері енергетики, оскільки вона обумовлює:

- зростання навантаження в зимовий період;
- підвищення попиту в літній період унаслідок кондиціювання повітря;
- добову та погодинну нерівномірність споживання [3].

Циклічна компонента

На відміну від сезонності, циклічність не має сталого періоду. Вона часто пов’язана з макроекономічними факторами, такими як економічні кризи, енергетичні кризи, зміни у тарифній політиці. Часто цикл триває кілька років, і точне визначення його меж є складним завданням.

Випадкова компонента

Це **шум або залишок**, що не може бути пояснений жодною з попередніх компонент. Він включає в себе:

- вимірювальні похибки;
- спонтанні відхилення;
- короткочасні збурення системи.

Для ефективного прогнозування необхідно, щоб залишки були **білі шумом** — мали нульове математичне сподівання та постійну дисперсію.

Вибір моделі представлення

Критерії вибору між адитивною та мультиплікативною моделлю:

- Якщо коливання не залежать від рівня ряду — адитивна модель;
- Якщо амплітуда коливань збільшується з ростом рівня — мультиплікативна модель [1].

У сфері прогнозування споживання електроенергії зазвичай застосовують **мультиплікативні моделі**, оскільки сезонні коливання (наприклад, піки взимку) мають більшу амплітуду при високих базових рівнях.

Детальний аналіз компонент дозволяє не лише будувати точні прогнози, але й **визначати моменти енергетичних піків**, які зазвичай пов’язані з **викидами надлишкової енергії у мережу**, що, своєю чергою, веде до додаткових навантажень на генераційні системи та екологічні наслідки [15]. Раціональне управління споживанням з урахуванням сезонності дозволяє мінімізувати **вплив на довкілля** через ефективніше балансування попиту й пропозиції.

1.3. Проблема стаціонарності та способи її подолання

Стаціонарність часового ряду — це властивість, за якої його статистичні характеристики не змінюються з часом. Формально, ряд вважається стаціонарним, якщо виконуються наступні умови:

1. Середнє значення ряду — стале:

$$\mathbb{E}[y_t] = \mu = \text{const}$$

2. Дисперсія — стала для всіх моментів часу:

$$\text{Var}(y_t) = \sigma^2 = \text{const}$$

3. Автоковаріація залежить лише від лага kkk, а не від моменту часу:

$$\mathrm{Cov}(y_t, y_{t-k}) = \gamma_k$$

Якщо ці три умови виконуються, ряд називають слабко стаціонарним або стаціонарним у широкому сенсі [1].

Більшість класичних статистичних моделей часових рядів (наприклад, ARIMA) вимагають, щоб вхідні дані були стаціонарними. Нестаціонарні ряди мають змінні тенденції (тренди), змінну амплітуду, які призводять до непередбачуваної поведінки моделі, збільшення похибки та неправильного навчання [2].

У контексті енергетики стаціонарність особливо важлива, адже нестабільні ряди можуть виникати під впливом:

- сезонних коливань попиту,
- зміни погодних умов,
- соціальних та економічних подій (наприклад, різкий спад у період локдаунів) [3].

Найбільш поширені способи діагностики стаціонарності:

1. Графічний аналіз — побудова часового графіка. Якщо середнє та дисперсія змінюються з часом — ряд нестаціонарний.
2. ACF/PACF-графіки — автокореляційні та часткові автокореляційні функції. Для стаціонарного ряду значення ACF швидко зменшується.
3. ADF-тест (Augmented Dickey–Fuller) — статистичний тест, що перевіряє гіпотезу про наявність одиничного кореня (ознака нестаціонарності).
4. KPSS-тест — навпаки, перевіряє гіпотезу про стаціонарність (у противагу ADF).

Методи стабілізації часових рядів

Якщо часовий ряд є нестаціонарним, його необхідно трансформувати до стаціонарного вигляду. Основні методи:

Диференціювання

Це найбільш поширений підхід у моделюванні ARIMA. Різниця між поточним та попереднім значенням допомагає усунути тренд:

$$y'_t = y_t - y_{t-1}$$

У загальному випадку можна виконати d-разове диференціювання:

$$y_t^{(d)} = \Delta^d y_t$$

де Δ — оператор різниці. Якщо після першого диференціювання ряд залишається нестаціонарним, можна застосувати друге тощо.

Логарифмування

Застосовується для стабілізації **дисперсії**:

$$y'_t = \log(y_t)$$

Після цього також можливе диференціювання, що дає **лог-різницю**:

$$y''_t = \log(y_t) - \log(y_{t-1}) = \log\left(\frac{y_t}{y_{t-1}}\right)$$

Цей підхід зручний для рядів, у яких зміни залежать від масштабу.

Згладжування (smoothing)

Мета — зменшити випадкові коливання. Наприклад, ковзне середнє:

$$\tilde{y}_t = \frac{1}{k}\sum_{i=0}^{k-1} y_{t-i}$$

Цей метод не забезпечує стаціонарності, але допомагає візуально оцінити тренд.

Віднімання тренду або сезонної складової

Метод декомпозиції дозволяє виокремити тренд і сезонність, а потім — працювати з залишковою серією. Послідовне застосування трансформацій дозволяє підготувати дані до побудови прогнозних моделей (рис.1.4)

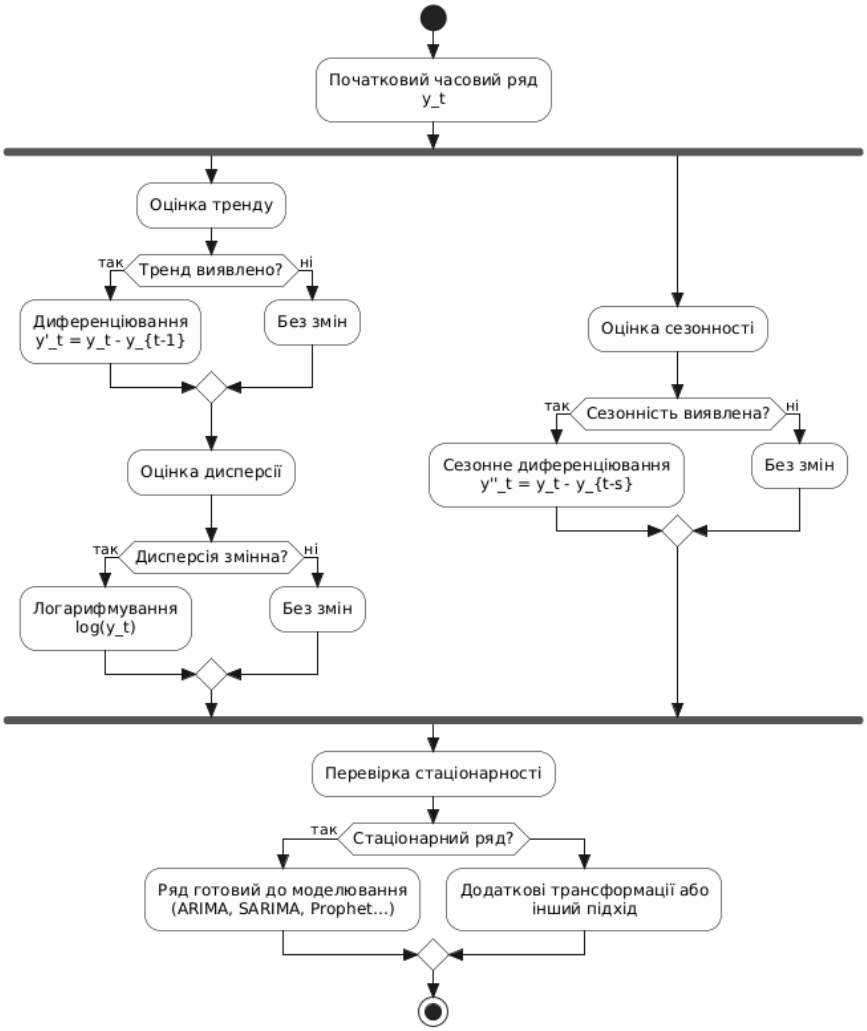

Рисунок 1.4 — Схема стабілізації часового ряду.

Роль стаціонарності в прогнозуванні енергоспоживання

У задачах прогнозування споживання електроенергії, нестаціонарність є типовим явищем, що виникає через:

- добову сезонність,
- поступовий ріст попиту,
- вплив погодних умов.

Коректне попереднє опрацювання даних дозволяє:

- уникнути псевдокореляцій,
- покращити точність прогнозу,

- зменшити похибку в періодах пікового навантаження [6].

Крім того, стаціонарні моделі краще інтерпретуються, що важливо при прийнятті рішень у системах енергоменеджменту та екологічного моніторингу [15].

1.4. Основні моделі прогнозування часових рядів

У практиці прогнозування часових рядів використовується широкий спектр моделей — від класичних статистичних до сучасних нейронних. Найбільш поширеними для уні- і мультиваріантних часових рядів є моделі ARIMA, її розширення SARIMA, а також методи експоненціального згладжування. Усі вони ефективно застосовуються в задачах, де потрібне коротко- або середньострокове передбачення на основі історичних даних [1], [6].

1.4.1. Модель ARIMA

Модель **ARIMA (AutoRegressive Integrated Moving Average)** є однією з базових у класичному підході до прогнозування часових рядів. Вона складається з трьох компонент:

- AR(p) — авторегресійна частина (враховує залежність поточного значення від попередніх);
- I(d) — інтегрування, або кількість диференціювань для досягнення стаціонарності;
- MA(q) — компонент ковзного середнього (враховує шумові коливання попередніх періодів).

Загальна формула ARIMA(p, d, q):

$$y'_t = c + \sum_{i=1}^{p} \phi_i y'_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

де:

y'_t — диференційований часовий ряд,

ϕ_i — коефіцієнти авторегресії,

θ_j — коефіцієнти ковзного середнього,

ε_t — білий шум.

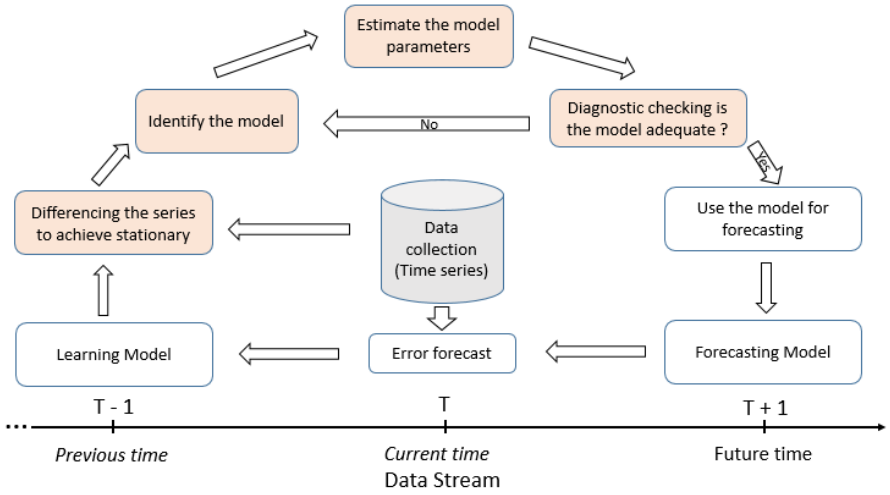

Рисунок 1.5 — Схематична структура ARIMA-моделі

Переваги ARIMA:

- ефективна для стаціонарних рядів після диференціювання;
- проста реалізація;
- дозволяє будувати точні короткострокові прогнози [2].

Недоліки:

- не враховує сезонність;
- складно інтерпретувати при високих значеннях p,d,q;
- не враховує зовнішні фактори (наприклад, температуру чи політику тарифів).

1.4.2. Модель SARIMA

SARIMA (Seasonal ARIMA) — це розширення моделі ARIMA, що дозволяє моделювати **сезонні закономірності**, характерні для багатьох

прикладних задач, зокрема — в енергетиці. SARIMA додає до ARIMA ще чотири параметри: P,D,Q,m, які відповідають за сезонні лаги.

Формат моделі:

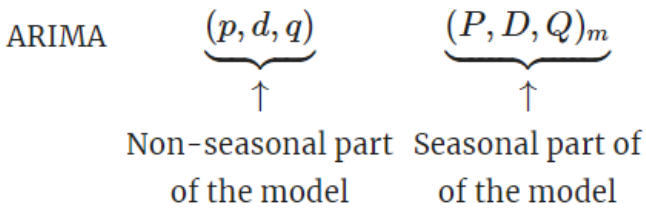

Рисунок 1.6 — Структура SARIMA-моделі

Переваги SARIMA:

- враховує сезонність;
- добре працює з річними, квартальними або добовими шаблонами;
- підтримується багатьма бібліотеками (наприклад, statsmodels у Python).

Недоліки:

- складніша ідентифікація параметрів;
- обмежена інтерпретованість при високих порядках;
- чутливість до вибору сезонного періоду.

1.4.3. Метод експоненціального згладжування (Holt-Winters)

Цей метод використовує **експоненціальне зважження минулих значень**, що дозволяє моделі швидше реагувати на останні зміни в ряді. Найвідоміші моделі:

- **Single exponential smoothing** — для рядів без тренду та сезонності;
- **Double exponential (Holt)** — для рядів із трендом;
- **Triple exponential (Holt-Winters)** — для рядів із трендом і сезонністю.

Переваги:

- гнучка модель для даних із сезонністю;
- адаптується до змін у тенденціях;

- підходить для автоматичних систем контролю та прогнозування.

Недоліки:

- чутливість до початкових умов;
- менш точна при складних структурах коливань;
- не враховує зовнішні чинники.

Розглянуті моделі — **ARIMA, SARIMA та Holt-Winters** — утворюють основу класичного аналізу часових рядів. Вони дозволяють моделювати тренди, сезонність та шумові коливання. У практиці прогнозування електроспоживання кожна з них має свої переваги залежно від структури даних. У подальших розділах буде обґрунтовано вибір оптимального підходу для розв'язання поставленої задачі.

1.5. Застосування моделей для прогнозування енергоспоживання

У галузі енергетики точне прогнозування попиту на електроенергію має вирішальне значення для збалансованої роботи енергосистеми, підвищення енергоефективності та скорочення шкідливих викидів у довкілля. Невірно спрогнозоване споживання може призводити до перевиробництва, аварійного відключення генераторів, перевантажень мережі та втрат енергії, що, своєю чергою, погіршує екологічну ситуацію [3], [15].

Споживання електроенергії формується під впливом широкого спектру факторів:

- Добова, тижнева та сезонна періодичність (наприклад, піки навантаження ввечері або в опалювальний сезон);
- Погодні умови (температура, вологість, хмарність) [5];
- Тарифна політика та поведінка споживачів (нічний тариф, промислові навантаження);

- Економічні та соціальні події (свята, надзвичайні ситуації).

Такі ряди є високоволатильними, нестаціонарними та часто мають сезонну структуру, що робить їх ідеальними кандидатами для моделей SARIMA або Holt-Winters [6].

У проектах із передбачення короткострокового (від кількох годин до кількох днів) споживання електроенергії, моделі ARIMA та SARIMA демонструють добрі результати, особливо при попередньому очищенні та диференціюванні ряду [1].

Зокрема:

- ARIMA підходить для прогнозу загального навантаження у відносно стабільних умовах без вираженої сезонності.
- SARIMA є ефективною у випадках з чітко вираженою сезонністю (наприклад, щотижневі або річні цикли), як це часто спостерігається у житловому секторі.

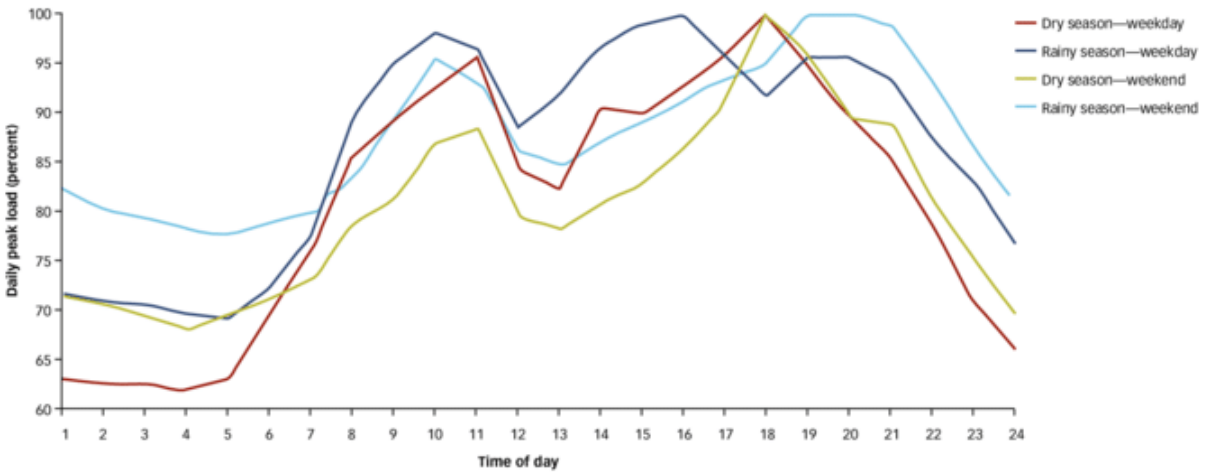

Рисунок 1.7 — Схематичний добовий профіль споживання електроенергії.

Використання моделі Prophet

Facebook Prophet — сучасна модель прогнозування, яка базується на адитивній композиції компонент:

$$y(t)=g(t)+s(t)+h(t)+\varepsilon_t$$

де:

$g(t)$ — тренд,

$s(t)$ — сезонність,

$h(t)$ — ефекти свят та подій,

ε_t — залишковий шум.

Переваги Prophet:

- автоматичне виявлення сезонності;
- підтримка святкових ефектів;
- зручний інтерфейс та інтеграція з Pandas;
- ефективність при роботі з пропущеними даними або нерівномірними спостереженнями [10].

У проектах, пов’язаних з енергоменеджментом або побудовою **систем попередження перевантажень**, Prophet дозволяє будувати інтерпретовані прогнози та включати зовнішні змінні (наприклад, температуру повітря) [9].

Глибокі нейромережеві підходи

У разі складної поведінки часового ряду або великої кількості зовнішніх змінних можуть застосовуватись **нейромережеві моделі**:

- LSTM (Long Short-Term Memory) — рекурентна мережа, здатна запам’ятовувати довгі послідовності;
- GRU (Gated Recurrent Unit) — компактна альтернатива LSTM;
- Transformer-based моделі — добре працюють із сезонними шаблонами [12].

Такі моделі демонструють високу точність, але потребують великих обсягів навчальних даних, часу на обчислення, а також ретельного налаштування гіперпараметрів [8].

Таблиця 1.1 – Порівняння класичних і сучасних підходів

Характеристика	**ARIMA/SARIMA**	**Prophet**	**LSTM/Transformer**
Сезонність	SARIMA: ✓	✓ (автоматично)	✓ (навчається)
Робота з трендом	✓	✓	✓

Характеристика	ARIMA/SARIMA	Prophet	LSTM/Transformer
Робота з подіями	✗	✓	частково
Чутливість до шуму	висока	середня	низька
Вимоги до даних	середні	низькі	високі
Пояснюваність результатів	✓	✓	✗

Моделі часових рядів активно використовуються в:

- платформах розумного дому (smart home) — для оптимізації використання техніки;
- промислових енергосистемах — для планування навантаження;
- системах диспетчеризації — прогнозування пікових періодів і балансування енергомережі;
- екологічному моніторингу — прогнозування надмірного споживання як джерела забруднення [4].

Ефективне прогнозування дозволяє:

- уникати перевиробництва електроенергії;
- зменшити викиди CO_2 завдяки точному плануванню;
- сприяти впровадженню відновлюваних джерел енергії, які потребують стабільного графіку навантажень [15].

Зокрема, дослідження [7] доводять, що використання машинного навчання у прогнозах енергоспоживання дозволяє зменшити втрати генерації на 8–15% та знизити пікові навантаження на 12–20%.

Таким чином, у результаті аналізу встановлено, що SARIMA та Prophet є найбільш перспективними моделями для задачі прогнозування споживання електроенергії з урахуванням сезонних та трендових характеристик. У подальших розділах буде обґрунтовано вибір відповідного інструментарію, реалізовано прототип моделі та здійснено її оцінку на основі реальних даних.

2. ОБГРУНТУВАННЯ ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ

2.1. Вимоги до інструментального середовища

Реалізація системи прогнозування споживання електроенергії на основі часових рядів вимагає відповідного програмного та апаратного забезпечення, що забезпечує ефективну роботу з даними, моделювання, збереження результатів і подальшу візуалізацію.

У цьому проєкті передбачено розробку консольного застосунку, який функціонуватиме на локальній машині. Такий підхід дозволяє:

- забезпечити автономність системи,
- мінімізувати залежність від зовнішніх платформ,
- отримати повний контроль над середовищем виконання,
- легко адаптувати застосунок під різні ОС (Windows, Linux, MacOS).

Таблиця 2.1 — Основні функціональні вимоги до середовища:

Вимога	**Пояснення**
Робота з табличними даними	Обробка CSV-файлів (час, дата, споживання, температура тощо)
Побудова моделей прогнозування	Моделювання на основі ARIMA, SARIMA, Prophet
Гнучке налаштування параметрів	Можливість змінювати кількість лагів, тип сезонності, довжину прогнозу
Візуалізація	Побудова графіків споживання, помилок, сезонних компонентів
Збереження результатів	Запис прогнозів у CSV-файл (з мітками часу)
Універсальність	Запуск на більшості систем із попередньо встановленим Python 3.x

- Технічні вимоги:
- Операційна система: Windows / Linux / MacOS (кросплатформенність)
- Інтерпретатор: Python 3.8+
- Формат вхідних/вихідних даних: .csv

- Обсяг даних: до 10–50 тис. записів (типовий розмір піврічного годинного ряду)
- Вимоги до оперативної пам’яті: ≥ 4 ГБ
- Платформа візуалізації: Matplotlib / Seaborn

Особливості реалізації:

- Застосунок працюватиме у діалоговому режимі в терміналі.
- Користувач матиме змогу:
    - вказати шлях до CSV-файлу;
    - обрати модель (ARIMA, SARIMA, Prophet);
    - задати параметри прогнозування;
    - зберегти результати у вигляді таблиці й графіків.

Попри те, що система реалізується як автономна локальна утиліта, її структуру передбачено у форматі, придатному до:

- інтеграції у веб-сервіси (через API),
- розгортання у хмарному середовищі (Google Colab, JupyterHub),
- підключення до баз даних або розумних лічильників у майбутньому.

Такий підхід узгоджується з сучасними тенденціями у сфері енергоменеджменту, що передбачають гнучке масштабування інструментів і можливість підключення до зовнішніх джерел даних [15].

2.2. Вибір мови програмування та середовища розробки

У контексті реалізації системи прогнозування енергоспоживання критично важливо підібрати інструмент, який забезпечує:

- високу гнучкість обробки даних;
- наявність бібліотек для статистичного та машинного навчання;
- зручність роботи з часовими рядами;
- можливості візуалізації та автоматизації процесів.

На основі аналізу сучасних мов програмування та середовищ розробки, а також враховуючи популярність інструментів у науковій спільноті, у цьому проєкті було обрано мову програмування Python.

Python — мова з відкритим вихідним кодом, яка активно використовується в науково-дослідницькому та прикладному програмуванні. Основні переваги її застосування в задачах прогнозування часових рядів:

Таблиця 2.2 — Критерії вибору мови програмування для проекту:

Критерій	**Характеристика Python**
Підтримка статистичних методів	statsmodels, scikit-learn, prophet
Обробка часових рядів	pandas, numpy, datetime
Побудова графіків	matplotlib, seaborn, plotly
Робота з файлами (CSV)	pandas.read_csv, csv, openpyxl
Гнучкість моделювання	Підтримка ARIMA, SARIMA, LSTM, Prophet
Простота реалізації CLI	Вбудовані засоби для створення консольних інтерфейсів (argparse)
Широке співтовариство	Багато готових рішень, прикладів і документації
Кросплатформеність	Працює на всіх популярних ОС без змін у коді

Таблиця 2.3 — Порівняння Python з альтернативними варіантами

Критерій	**Python**	**R**	**MATLAB**
Обробка даних	✓✓✓	✓✓✓	✓✓
Прогнозування	✓✓✓	✓✓✓	✓✓
Консольна утиліта	✓✓✓	✗	✗
Візуалізація	✓✓✓	✓✓✓	✓✓
Безкоштовність	✓✓✓	✓✓✓	✗ (платна ліцензія)
Впровадження у системи	✓✓✓	✓	✗
Модель гібридного прогнозу	✓✓✓ (LSTM + ARIMA)	частково	частково

Висновок: хоча R має сильні статистичні можливості, він менш придатний для розгортання **утиліт та застосунків**, що важливо для цього проєкту. MATLAB обмежений ліцензією й призначений радше для лабораторних задач, а не для практичної реалізації в open-source середовищі [14].

Обране середовище розробки

Для реалізації та тестування обрано локальне середовище з інтерпретатором Python 3.10+, а також один із наведених інструментів:

- Visual Studio Code (із розширеннями Python, Jupyter, Pylance)
- PyCharm Community Edition — як альтернатива для більш зручної навігації у великих проєктах
- Jupyter Notebook — для експериментального аналізу та перевірки гіпотез

Завдяки широкому спектру бібліотек, Python дозволяє легко перейти від дослідницького аналізу до повноцінного програмного продукту, включаючи:

- інтеграцію через API,
- використання в хмарі (Google Colab),
- створення графічних інтерфейсів (Tkinter, Streamlit),
- підключення до баз даних (SQLite, PostgreSQL).

2.3. Бібліотеки для обробки даних і моделювання часових рядів

Для реалізації системи прогнозування споживання електроенергії необхідно забезпечити:

- передобробку даних (очищення, трансформації, агрегація);
- аналіз часових рядів (перевірка стаціонарності, автокореляція);
- побудову прогнозних моделей (ARIMA, SARIMA, Prophet);
- порівняння результатів та метрики точності (MAE, RMSE);
- гнучку інтеграцію з файлами CSV;
- візуалізацію результатів прогнозу.

На основі цих вимог, до складу обраного технологічного стеку включено такі ключові бібліотеки:

Pandas для обробки табличних даних, агрегацій, роботи з часовими індексами

Переваги:

- підтримка datetime-індексів;
- гнучке зчитування та збереження .csv;
- вбудовані функції ресемплінгу, згладжування, зміщення.

```
import pandas as pd
df = pd.read_csv("data/input.csv", parse_dates=["datetime"],
index_col="datetime")
df = df.resample("H").mean()  # годинне агрегування
```

NumPy для математичних операцій, диференціювання, генерації масивів. Використовується у більшості моделей як базова структура

StatsModels для реалізації класичних статистичних моделей: AR, MA, ARIMA, SARIMA

Можливості:

- побудова моделей з ручним або автоматичним підбором параметрів;
- перевірка стаціонарності (ADF, KPSS);
- інтерпретовані результати: коефіцієнти, p-значення, автокореляції;
- вбудовані методи для побудови графіків залишків.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(df['value'], order=(1,1,1),
seasonal_order=(1,1,1,24))
result = model.fit()
forecast = result.predict(start="2024-01-01", end="2024-01-07")
```

Prophet (Meta/Facebook) Побудова адитивних моделей: тренд + сезонність + свята

Переваги:

- автоматичне виявлення сезонних циклів;
- простота використання (API у стилі sklearn);
- підтримка відсутніх значень, нерівномірних інтервалів;
- зручне візуальне представлення результатів.

from prophet import Prophet

```
df_prophet = df.reset_index().rename(columns={"datetime": "ds",
"value": "y"})
model = Prophet(daily_seasonality=True)
model.fit(df_prophet)
future = model.make_future_dataframe(periods=48, freq="H")
forecast = model.predict(future)
```

Scikit-learn для реалізації метрик оцінки моделей:

- MAE (mean absolute error)
- MSE (mean squared error)
- RMSE (root mean squared error)

```
from sklearn.metrics import mean_squared_error
import numpy as np

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
```

Альтернативи та розширення

LSTM (Long Short-Term Memory)

Моделі LSTM (на базі TensorFlow або PyTorch) можуть бути використані на наступному етапі для побудови нейронних прогнозів. Однак вони мають вищу складність реалізації та потребують великих обсягів даних для якісного навчання [8].

Таблиця 2.4 — Порівняння бібліотек для прогнозування

Бібліотека	**Модель**	**Сезонність**	**Врахування свят**	**Пропущені значення**	**Інтерпретованість**	**Прогнозування**
StatsModels	ARIMA/SARIMA	ручне налаштування	✗	✗	✓✓✓	✓✓✓
Prophet	Prophet	автоматично	✓	✓	✓✓	✓✓✓
TensorFlow	LSTM	автоматично	✗	частково	✗	✓✓✓

Висновок: для поточного проєкту рекомендовано використовувати StatsModels для класичної SARIMA-моделі та Prophet для порівняння результатів, оскільки вони добре працюють з часовими рядами середнього

розміру та не потребують глибокого налаштування нейронних архітектур [6], [10], [14].

2.4. Інструменти для візуалізації результатів

Візуалізація є невід'ємною частиною системи прогнозування енергоспоживання, оскільки вона:

- дозволяє наочно представити динаміку реального та прогнозованого споживання;
- демонструє ефективність моделі (відповідність, залишки, помилки);
- сприяє прийняттю рішень користувачами або адміністраторами енергосистем;
- допомагає виявити аномалії або повторювані шаблони.

Для реалізації візуалізації обрано бібліотеки **Matplotlib** і **Seaborn** — найбільш гнучкі, зручні та сумісні з іншими компонентами Python-стека.

Matplotlib — базова бібліотека для побудови графіків, яка дозволяє створювати високоякісні статичні, інтерактивні та анімовані візуалізації.

Основні переваги:

- підтримка масштабування, форматування осей, підписів;
- можливість побудови багатокомпонентних графіків (напр. фактичне vs прогнозоване споживання);
- збереження графіків у форматах PNG, SVG, PDF.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(df.index, df['value'], label='Фактичне')
plt.plot(forecast.index, forecast['yhat'], label='Прогноз',
linestyle='--')
plt.xlabel("Час")
plt.ylabel("Споживання (кВт·год)")
plt.title("Прогнозування споживання електроенергії")
```

```
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("graphs/forecast.png")
plt.show()
```

Seaborn — розширення для Matplotlib з підтримкою статистичних графіків і привабливішим стилем оформлення.

Застосовується для:

- аналізу сезонних коливань;
- візуалізації кореляцій та автокореляцій;
- побудови boxplot / heatmap / lineplot.

```
import seaborn as sns

sns.set(style="whitegrid")
sns.lineplot(data=df, x="datetime", y="value")
plt.title("Динаміка споживання")
```

Модель **Prophet** має вбудовані засоби візуалізації компонентів:

```
from prophet.plot import plot_components_plotly
plot_components_plotly(model, forecast)
```

Це дозволяє окремо побачити:

- тренд;
- денну/тижневу/річну сезонність;
- ефекти свят (якщо задано).

Таблиця 2.5 — Альтернативні засоби візуалізації

Бібліотека	**Характеристика**	**Недоліки у нашому контексті**
Plotly	Інтерактивні графіки у браузері	Потребує більше налаштувань, не завжди зручно у консолі
Bokeh	Потужна бібліотека для web-візуалізації	Надмірна для локального консольного застосунку
Altair	Декларативний стиль, добре інтегрується з pandas	Менша підтримка складних кастомних графіків

2.5. Засоби збору та зберігання даних

Для реалізації системи прогнозування споживання електроенергії на основі часових рядів необхідним є ефективний механізм збору, зберігання та підготовки вхідних даних. Основні вимоги до даних:

- наявність часових міток (datetime),
- регулярний інтервал спостережень (щогодинний або щоденний),
- відсутність пропусків або коректна їх обробка,
- зручність завантаження та подальшого аналізу.

Таблиця 2.6 — Типи даних, що використовуються у проекті

Джерело	Формат	Опис
Дані розумних лічильників	CSV	Споживання у кВт·год з часовими мітками
Метеодані (температура тощо)	CSV	Допоміжні змінні для покращення точності моделі
Публічні відкриті набори	CSV	Дані зі сайтів типу Open Power System Data або EnergyPlus [5]
Згенеровані дані (тестові)	CSV	Для валідації моделі на контрольованих прикладах

Вибір **CSV (Comma-Separated Values)** як основного формату зберігання обґрунтовується такими перевагами:

- легкий та широко підтримуваний формат;
- сумісність з pandas, Excel, будь-яким текстовим редактором;
- простота обробки при великому обсязі даних;
- не потребує встановлення СУБД.

Таблиця 2.6 — Рекомендована структура CSV-файлу

datetime	consumption	temperature	day_of_week
2024-01-01 00:00:00	4.23	-3.1	Monday
2024-01-01 01:00:00	3.98	-3.2	Monday
...	...	...	...

Значення datetime повинні мати формат ISO (YYYY-MM-DD HH:MM:SS)

Колонка consumption — основна змінна прогнозування

Додаткові змінні (temperature, day_of_week) можуть використовуватись як регресори у Prophet

Обробка пропущених значень

Пропущені значення можуть виникати через: збій у роботі лічильника, пропуски в переданих даних, збій під час збору з API.

Можливі стратегії обробки: Лінійна інтерполяція, Заповнення середнім значенням, Видалення пропущених рядків

Зберігання результатів

Результати прогнозу зберігаються також у форматі **CSV**, з окремим стовпчиком forecast і часовою міткою. Це дозволяє:

- будувати графіки (співставлення реальне vs прогнозоване);
- використовувати результати у звітах та презентаціях;
- передавати дані на вхід інших модулів (напр., web API або інтеграція у SCADA).

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ЗАСТОСУНКУ

3.1. Вимоги до моделі прогнозування

Модель прогнозування споживання електроенергії, що розробляється в рамках роботи, є центральним компонентом системи прийняття рішень щодо керування енергоресурсами. Вона повинна враховувати сезонні, добові та трендові особливості змін у навантаженні, а також мати здатність адаптуватися до різних обсягів вхідних даних.

Основна мета моделі: побудова точного коротко- або середньострокового прогнозу споживання електроенергії на основі часових рядів з урахуванням трендових і сезонних коливань, а також зовнішніх факторів (погода, доба тижня).

Функціональні вимоги

- Модель повинна приймати вхідні дані у форматі CSV (із часовою міткою).
- Модель має здійснювати обробку пропущених значень та перевірку стаціонарності.
- Модель повинна автоматично або вручну підбирати параметри (напр. для ARIMA/SARIMA).
- Повинна підтримуватись можливість використання допоміжних регресорів (наприклад, температура).
- Результати прогнозу повинні зберігатись у форматі CSV та виводитись графічно.
- Користувач повинен мати змогу задати тривалість прогнозу (наприклад, 24, 48, 168 годин).

Нефункціональні вимоги

- Прогнозування має виконуватись на звичайному персональному комп’ютері (≥ 4 ГБ ОЗП).
- Система повинна забезпечити обробку даних обсягом до 50 000 записів.

- Прогноз має генеруватися протягом ≤ 10 секунд для одного тижня даних.
- Модель має демонструвати RMSE ≤ 10% від середнього навантаження (для годинних даних).
- Код та структура мають бути уніфікованими та придатними до масштабування (наприклад, розширення на щохвилинні дані).

Таблиця 3.1 — Вхідні дані

Поле	Тип	Опис
datetime	datetime	Часова мітка
consumption	float	Споживання електроенергії (кВт·год)
temperature	float	Температура повітря (°C) – опціонально
day_of_week	string/int	День тижня

Таблиця 3.2 — Вихідні дані

Поле	Тип	Опис
datetime	datetime	Часова мітка прогнозованого значення
forecast	float	Прогнозоване споживання (кВт·год)
lower_bound	float	Нижня межа довірчого інтервалу (якщо підтримується)
upper_bound	float	Верхня межа довірчого інтервалу

Очікувані характеристики моделі:

- Адаптивність: підлаштовується під нові тренди та сезонні зміни.
- Гнучкість: підтримка кількох алгоритмів (SARIMA, Prophet) — можлива подальша інтеграція LSTM.
- Простота взаємодії: консольний режим з діалоговим інтерфейсом.
- Інтерпретованість: модель повинна дозволяти аналізувати вплив сезонності, тренду, зовнішніх чинників.

3.2 Архітектура системи прогнозування

Проєктування архітектури системи прогнозування споживання електроенергії базується на принципах модульності, незалежності підсистем і

розширюваності. Така структура дозволяє спростити супровід коду, змінювати окремі компоненти без впливу на інші, а також легко адаптувати систему до нових задач або джерел даних.

Основні підсистеми

Система умовно поділяється на п’ять функціональних модулів:

1. Консольний інтерфейс

- Діалог з користувачем;
- Введення параметрів: шлях до файлу, модель, період прогнозу тощо;
- Вивід повідомлень та інструкцій.

2. Модуль завантаження даних

- Зчитування даних із CSV-файлу;
- Перевірка коректності структури;
- Обробка пропущених значень;
- Формування базового DataFrame.

3. Модуль обробки даних

- Інтерполяція/очищення;
- Агрегація, ресемплінг (годинні/денні значення);
- Визначення сезонності, логарифмування, диференціювання;
- Розбиття на навчальний і тестовий набори.

4. Модуль моделювання

- Побудова SARIMA або Prophet-моделі;
- Навчання та валідація;
- Генерація прогнозу;
- Оцінка метрик (MAE, RMSE).

5. Модуль збереження та виводу результатів

- Експорт прогнозу у CSV;
- Побудова графіків фактичного та прогнозованого значення;
- Збереження результатів у директорію /graphs.

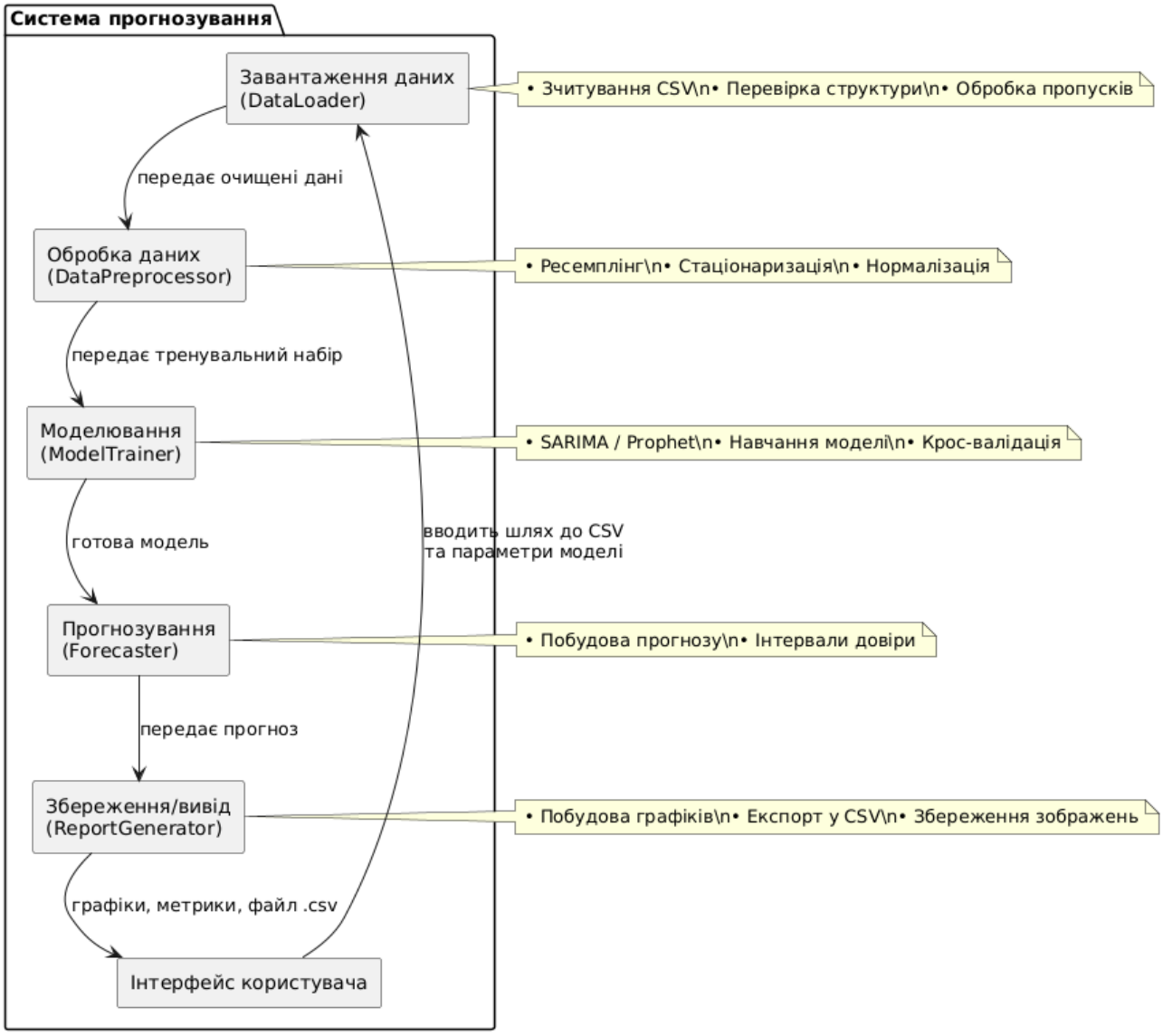

Рисунок 3.1 — Архітектура застосунку

- Стрілки вказують напрямок потоку даних;
- Компоненти ізольовані, кожен має одну відповідальність (принцип SRP);
- Ієрархічна структура дозволяє легко відлагоджувати й розширювати систему.

Архітектурні переваги

- Гнучкість: можна змінити модель (напр. замість SARIMA використати LSTM) без зміни решти системи;

- Масштабованість: легко додати нові джерела даних або API для онлайн-збірки;
- Інтегрованість: структура дозволяє в майбутньому створити API або графічний інтерфейс.

Діаграма потоків даних (DFD)

DFD дозволяє візуалізувати взаємозв'язки між підсистемами та основними потоками даних у межах системи прогнозування. Це допомагає:

- формалізувати структуру обробки даних;
- ідентифікувати ключові джерела, сховища та виходи;
- перевірити коректність інтеграції модулів.

На цьому рівні система розглядається як **"чорна скринька"**, що взаємодіє з зовнішніми суб'єктами.

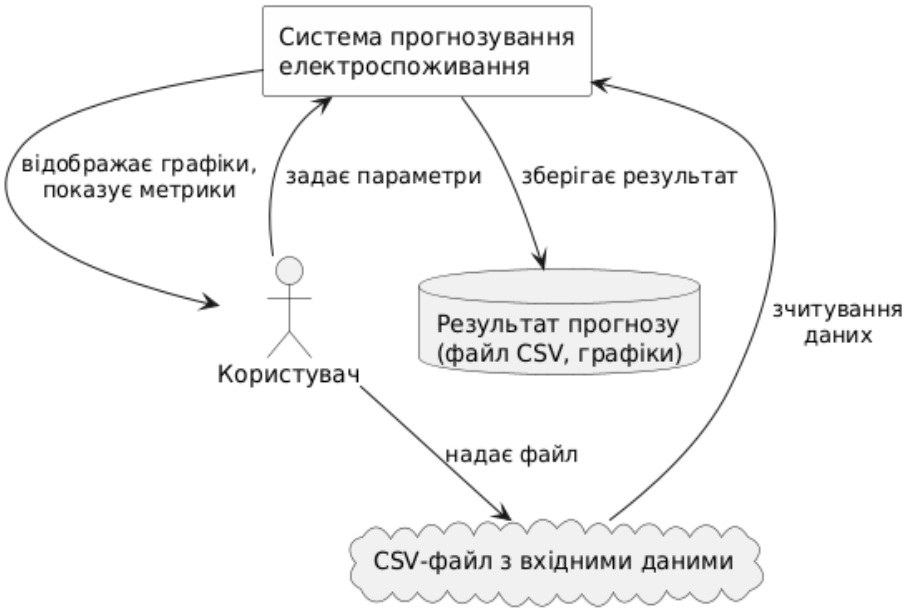

Рисунок 3.2 — Контекстна модель системи прогнозування

- Суб'єкт: Користувач (User) — взаємодіє з системою через CLI.
- Зовнішні джерела/сховища: вхідні дані у CSV, вихідні результати — також у CSV і графіках.
- Система: в центрі, як єдиний функціональний блок.

Діаграма класів застосунку

UML-діаграма класів необхідна для моделювання **внутрішньої об'єктної структури** системи, що реалізує функціонал прогнозування споживання електроенергії. Такий підхід дозволяє:

- чітко виділити логіку застосунку;
- реалізувати принципи SOLID;
- забезпечити масштабованість та підтримуваність коду.

Таблиця 3.3 — Основні класи системи:

Клас	**Опис функцій**
DataHandler	Завантаження, перевірка та очищення даних із CSV
Preprocessor	Стаціонаризація, ресемплінг, нормалізація
ModelBuilder	Ініціалізація та тренування моделей (SARIMA, Prophet)
Forecaster	Формування прогнозу на основі обраної моделі
MetricsEvaluator	Розрахунок MAE, MSE, RMSE
Visualizer	Побудова графіків: фактичне vs прогноз, залишки
CLIInterface	Робота з користувачем через консоль (введення параметрів, запуск моделей)

- Принцип єдності відповідальності (SRP): кожен клас має чітку функціональну роль;
- Інкапсуляція: поля — приховані, доступ через публічні методи;
- Гнучкість: легко підключати інші моделі або інтерфейси (наприклад, WebAPI).

Можливі розширення:

- окремі класи для роботи з погодними даними;
- підкласи для різних типів моделей (SARIMA, Prophet, LSTM);
- реалізація шаблонів проєктування (наприклад, Strategy для моделі).

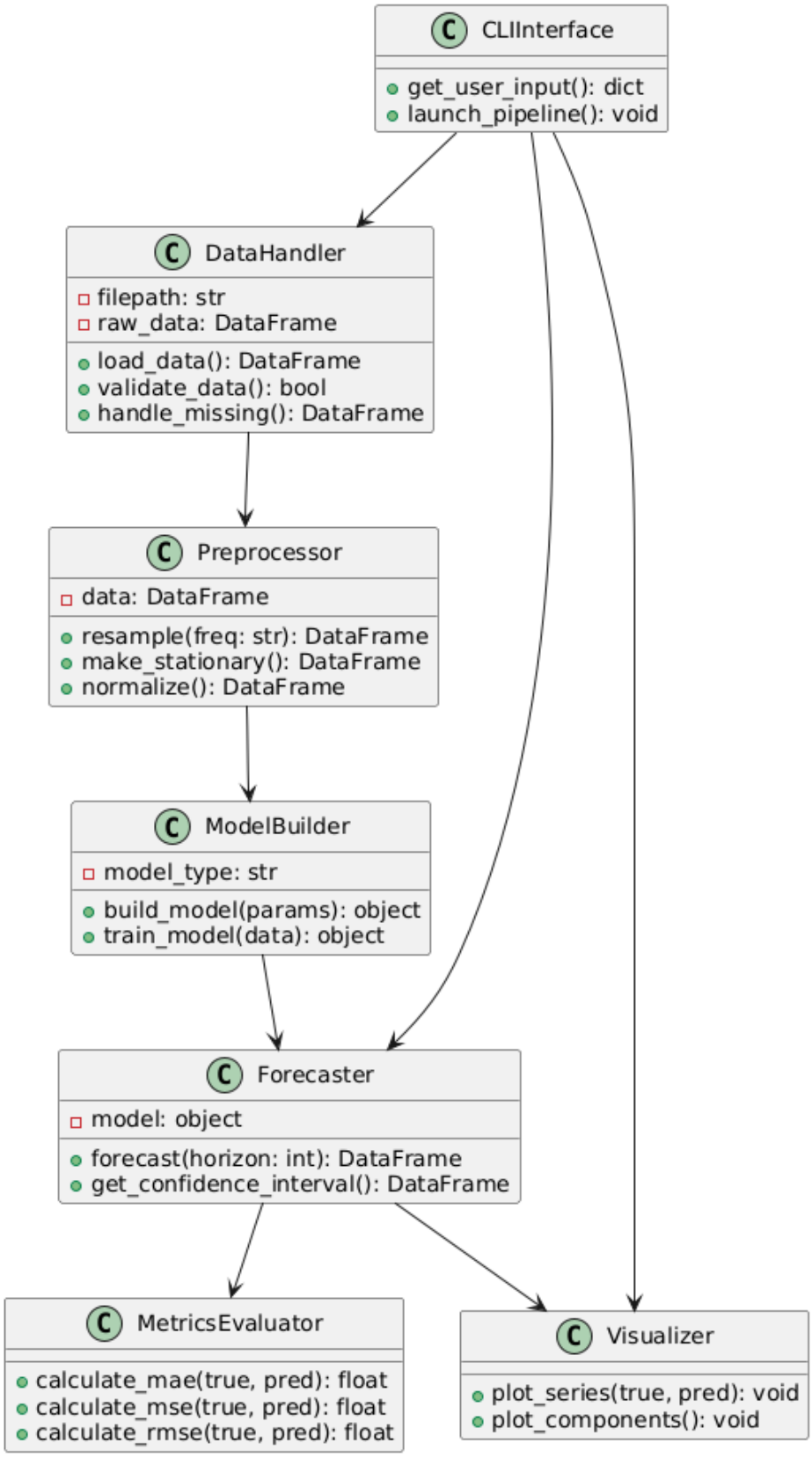

Рисунок 3.3 — Діаграма класів системи прогнозування

3.3 Алгоритм обробки часових рядів

Створення алгоритмічної блок-схеми необхідне для візуального представлення логіки виконання основного робочого процесу застосунку. Це забезпечує:

- ясність для розробників;
- спрощення майбутнього тестування;
- перевірку повноти реалізації вимог.

Основні етапи обробки:

1. Завантаження CSV-файлу з даними
2. Перевірка та очищення структури
3. Обробка пропущених значень
4. Перетворення до стаціонарного ряду (якщо потрібно)
5. Поділ даних на train/test
6. Побудова та тренування моделі
7. Прогнозування
8. Оцінка точності (MAE, MSE, RMSE)
9. Збереження прогнозу та візуалізація результатів

Алгоритм побудований з розгалуженням на етапах перевірки пропущених даних і стаціонарності, що відповідає кращим практикам моделювання часових рядів.

Враховано можливість вибору моделі та розділення набору даних для валідації.

Всі дії завершуються збереженням результату та побудовою графіків.

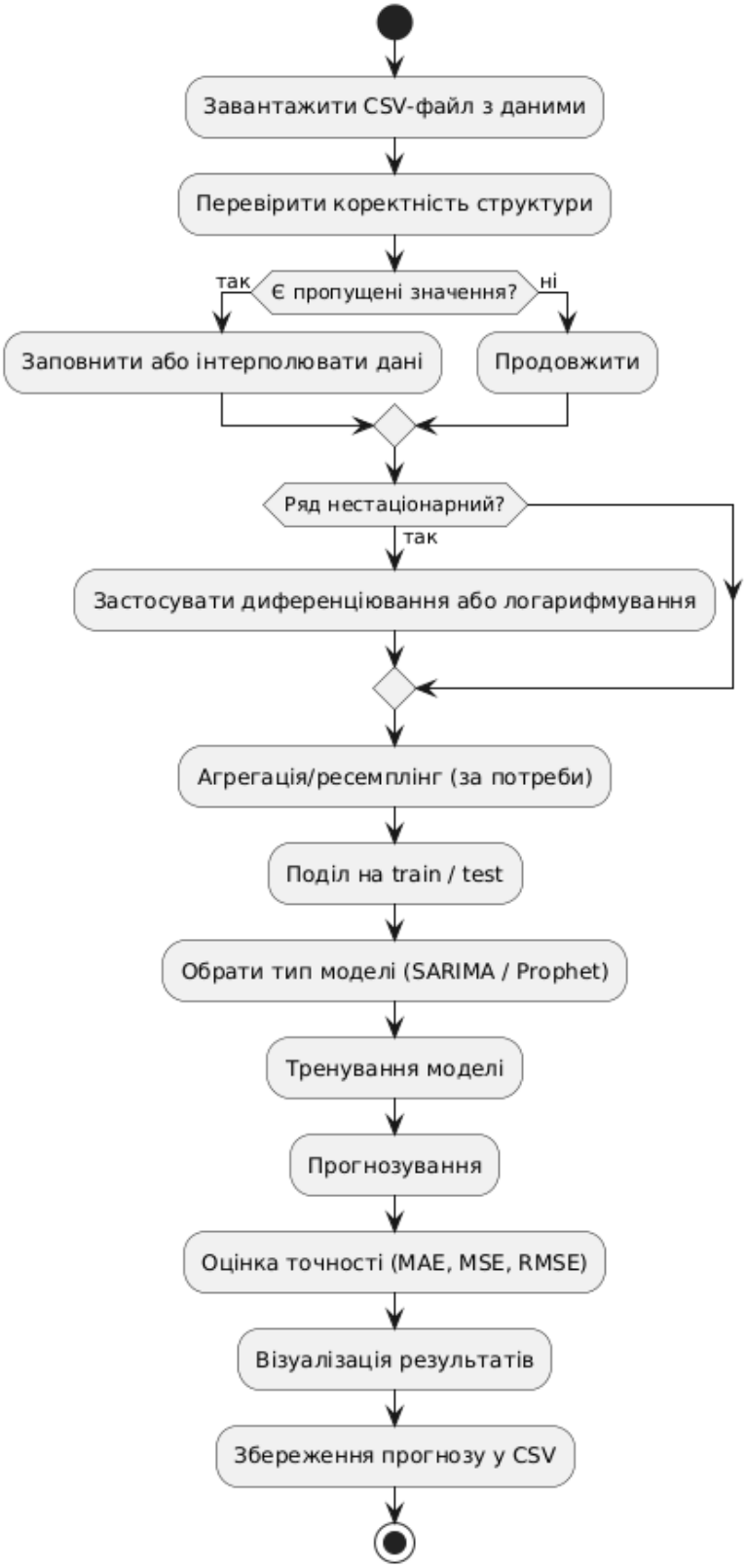

Рисунок 3.4 — Алгоритм роботи застосунку

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1. Структура програмної реалізації

Розроблений застосунок є консольною системою для прогнозування споживання електроенергії на основі часових рядів із використанням моделей SARIMA та Prophet. Він дозволяє автоматизовано обробляти CSV-дані, очищати пропущені значення, нормалізувати та стаціонаризувати ряди, будувати прогнози на заданий період, а також оцінювати якість прогнозу за допомогою стандартних метрик (MAE, MSE, RMSE). Результати зберігаються у вигляді графіків та таблиць.

Застосунок реалізовано у вигляді модульного Python-проєкту. Уся логіка розділена на окремі компоненти (модулі), які відповідають за різні етапи обробки даних. Користувач взаємодіє із системою через консоль, задаючи шлях до вхідного файлу, модель прогнозування та бажаний горизонт прогнозу. Вихідними є CSV-файл з прогнозом та візуалізація результатів.

```
energy_forecast/
├── data/                    ← Вхідні дані (CSV)
│   └── sample_data.csv
├── results/                 ← Результати прогнозу (CSV, графіки)
│   └── forecast_plot.png
├── models/                  ← Збережені моделі (якщо потрібно)
│   └── model_sarima.pkl
├── src/                     ← Основний код застосунку
│   ├── data_loader.py
│   ├── preprocessor.py
│   ├── model_sarima.py
│   ├── model_prophet.py
│   ├── visualizer.py
│   ├── evaluator.py
│   └── cli.py
├── main.py                  ← Точка входу в застосунок
├── requirements.txt         ← Список необхідних бібліотек
└── README.md                ← Інструкція по запуску
```

Рисунок 4.1 — Файлова структура проекту

data_loader.py

Модуль відповідає за завантаження вхідного CSV-файлу, перевірку наявності необхідних стовпців, приведення стовпця з датою до типу datetime і базове очищення пропущених значень.

```
# src/data_loader.py

import pandas as pd

class DataLoader:
    def __init__(self, filepath):
        self.filepath = filepath

    def load_data(self):
        try:
            data = pd.read_csv(self.filepath)
            if 'datetime' not in data.columns or 'consumption' not
in data.columns:
                raise ValueError("CSV must contain 'datetime' and
'consumption' columns.")
            data['datetime'] = pd.to_datetime(data['datetime'])
            data.set_index('datetime', inplace=True)
            data = data.sort_index()
            return data
        except Exception as e:
            raise RuntimeError(f"Error loading data: {e}")
```

preprocessor.py

Модуль для попередньої обробки даних: інтерполяція пропусків, ресемплінг за періодами (година, день), нормалізація та диференціювання для стаціонарності.

```
# src/preprocessor.py

import pandas as pd

class Preprocessor:
    def __init__(self, data):
        self.data = data
```

```
    def interpolate_missing(self):
        return self.data.interpolate(method='time')

    def resample(self, freq='H'):
        return self.data.resample(freq).mean()

    def difference(self):
        return self.data.diff().dropna()

    def normalize(self):
        return (self.data - self.data.mean()) / self.data.std()
```

model_sarima.py

Модуль реалізує побудову та тренування моделі SARIMA на основі бібліотеки statsmodels. Містить параметри сезонності та можливість прогнозування на заданий горизонт.

```
# src/model_sarima.py

from statsmodels.tsa.statespace.sarimax import SARIMAX

class SARIMAModel:
    def __init__(self, data, order=(1,1,1),
seasonal_order=(1,1,1,24)):
        self.data = data
        self.order = order
        self.seasonal_order = seasonal_order
        self.model = None
        self.results = None

    def train(self):
        self.model = SARIMAX(self.data, order=self.order,
seasonal_order=self.seasonal_order)
        self.results = self.model.fit(disp=False)

    def forecast(self, steps=24):
        return
self.results.get_forecast(steps=steps).predicted_mean
```

model_prophet.py

Модуль для побудови та запуску моделі Prophet. Підходить для рядів із сильно вираженою сезонністю. Підготовка даних до формату Prophet (ds, y) і прогнозування.

```
# src/model_prophet.py

from prophet import Prophet
import pandas as pd

class ProphetModel:
    def __init__(self, data):
        self.model = Prophet()
        self.data = data.reset_index().rename(columns={'datetime':
'ds', 'consumption': 'y'})

    def train(self):
        self.model.fit(self.data)

    def forecast(self, periods=24, freq='H'):
        future = self.model.make_future_dataframe(periods=periods,
freq=freq)
        forecast = self.model.predict(future)
        return forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

visualizer.py

Модуль для побудови графіків: фактичне vs прогноз, залишки, довірчі інтервали. Зберігає зображення у директорію results/.

```
# src/visualizer.py

import matplotlib.pyplot as plt

class Visualizer:
    def plot_forecast(self, actual, predicted, title="Прогноз vs
Реальні дані", save_path=None):
        plt.figure(figsize=(10,5))
        plt.plot(actual, label="Факт")
        plt.plot(predicted, label="Прогноз")
        plt.xlabel("Час")
        plt.ylabel("Споживання (кВт·год)")
```

```
        plt.title(title)
        plt.legend()
        if save_path:
            plt.savefig(save_path)
        plt.show()
```

evaluator.py

Модуль для обчислення метрик якості прогнозу: **MAE**, **MSE**, **RMSE**. Використовується для оцінки точності моделі на тестових даних.

```
# src/evaluator.py

from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

class Evaluator:
    def __init__(self, y_true, y_pred):
        self.y_true = y_true
        self.y_pred = y_pred

    def mae(self):
        return mean_absolute_error(self.y_true, self.y_pred)

    def mse(self):
        return mean_squared_error(self.y_true, self.y_pred)

    def rmse(self):
        return np.sqrt(self.mse())
```

cli.py

Інтерфейс командного рядка. Отримує параметри від користувача (файл, модель, горизонт), передає їх у відповідні модулі, організовує повний робочий цикл.

```
# src/cli.py

import argparse

def parse_args():
    parser = argparse.ArgumentParser(description="Energy
```

```
Consumption Forecasting Tool")
    parser.add_argument('--input', type=str, required=True,
help='Шлях до CSV з даними')
    parser.add_argument('--model', type=str, choices=['sarima',
'prophet'], required=True, help='Тип моделі')
    parser.add_argument('--period', type=int, default=24,
help='Горизонт прогнозу (у годинах)')
    parser.add_argument('--output', type=str,
default='results/forecast_plot.png', help='Шлях до графіка
прогнозу')
    return parser.parse_args()
```

4.2. Завантаження і попередня обробка даних

Для побудови достовірної моделі прогнозування необхідно забезпечити якісну підготовку вхідного набору даних. У цьому підрозділі розглядається повний pipeline: від завантаження CSV до отримання стаціонарного, очищеного ряду, готового до моделювання.

Формат вхідного CSV-файлу

Вхідний файл повинен містити щонайменше два стовпці:

datetime	**consumption**
2023-01-01 00:00:00	256.4
2023-01-01 01:00:00	242.1
...	...

Додаткові колонки (наприклад, температура) можуть бути використані в розширених версіях моделі, але є необов’язковими.

Приклад CSV (фрагмент)

```
datetime,consumption
2023-01-01 00:00:00,256.4
2023-01-01 01:00:00,242.1
2023-01-012:00:00,230.3
...
```

Основні кроки попередньої обробки

1. Завантаження та перевірка структури
2. Інтерполяція пропущених значень
3. Ресемплінг (за потреби)
4. Диференціювання для досягнення стаціонарності
5. Нормалізація для роботи моделей, чутливих до масштабів

Код робочого циклу попередньої обробки

```
from src.data_loader import DataLoader
from src.preprocessor import Preprocessor

# Крок 1: Завантаження даних
loader = DataLoader("data/sample_data.csv")
data = loader.load_data()

# Крок 2: Ініціалізація препроцесора
prep = Preprocessor(data)

# Крок 3: Інтерполяція пропущених значень
clean_data = prep.interpolate_missing()

# Крок 4: (Опційно) ресемплінг до годинного інтервалу
resampled_data = prep.resample(freq='H')

# Крок 5: Диференціювання (для SARIMA)
stationary_data = prep.difference()

# Крок 6: Нормалізація
normalized_data = prep.normalize()
```

Приклад результату (до/після інтерполяції)

```
До:
2023-01-01 00:00:00    256.4
2023-01-01 01:00:00       NaN
2023-01-01 02:00:00    230.3

Після:
2023-01-01 00:00:00    256.4
2023-01-01 01:00:00    243.35  ← інтерпольоване значення
```

```
2023-01-01 02:00:00    230.3
```

Переваги такого підходу

- Гнучкість: легко змінити частоту (наприклад, на денну — 'D');
- Можливість адаптації до обривів/випадкових пропусків;
- Уніфікований pipeline для обох моделей (SARIMA і Prophet).

4.3. Реалізація моделей прогнозування

Реалізація моделі SARIMA

Реалізація прогнозної моделі SARIMA відбувається в межах об’єктно-орієнтованої структури застосунку, описаної в розділах 3.4 та 4.1. У цьому підрозділі демонструється повний практичний pipeline побудови SARIMA-прогнозу з використанням очищених та нормалізованих даних.

Система організована так, щоб користувач взаємодіяв лише через консоль або функцію запуску (наприклад, main.py), задаючи вхідні параметри, а внутрішні компоненти (DataLoader, Preprocessor, SARIMAModel, Visualizer, Evaluator) послідовно обробляють дані, будують модель, створюють прогноз та оцінюють якість.

Особливу увагу було приділено коректному поділу даних на навчальний і тестовий набори (використовується 80/20), а також відповідному масштабуванню даних до застосування моделі та оцінки результатів. Прогноз будується на ту ж довжину, що й тестовий набір, для коректної валідації.

Результати зберігаються в каталозі results/, зокрема графіки, CSV-файли з прогнозом, а також значення метрик (MAE, MSE, RMSE), що можуть використовуватись для порівняння моделей.

Код реалізації SARIMA-прогнозу:

```
from src.data_loader import DataLoader
from src.preprocessor import Preprocessor
```

```
from src.model_sarima import SARIMAModel
from src.visualizer import Visualizer
from src.evaluator import Evaluator
import pandas as pd
import os

# Завантаження даних
loader = DataLoader("data/sample_data.csv")
raw_data = loader.load_data()

# Попередня обробка
prep = Preprocessor(raw_data)
cleaned_data = prep.interpolate_missing()
resampled_data = prep.resample(freq='H')          # або 'D' для
щоденних
stationary_data = prep.difference()
normalized_data = prep.normalize()

# Поділ на train / test
split_index = int(len(normalized_data) * 0.8)
train = normalized_data.iloc[:split_index]
test = normalized_data.iloc[split_index:]

# Ініціалізація та тренування SARIMA-моделі
sarima = SARIMAModel(
    data=train,
    order=(1, 1, 1),
    seasonal_order=(1, 1, 1, 24)  # добовий цикл (24 години)
)
sarima.train()

# Прогнозування на довжину тестового набору
forecast = sarima.forecast(steps=len(test))

# Збереження результатів прогнозу в CSV
forecast_df = pd.DataFrame({
    "datetime": test.index,
    "actual": test.values.flatten(),
    "forecast": forecast.values
})
os.makedirs("results", exist_ok=True)
forecast_df.to_csv("results/sarima_forecast.csv", index=False)
```

```
# Візуалізація результату
viz = Visualizer()
viz.plot_forecast(
    actual=test,
    predicted=forecast,
    title="SARIMA: Фактичні vs Прогнозовані значення",
    save_path="results/sarima_forecast.png"
)

# Оцінка точності
evaluator = Evaluator(test.squeeze(), forecast)
print("Оцінка точності моделі SARIMA:")
print("MAE:", round(evaluator.mae(), 4))
print("MSE:", round(evaluator.mse(), 4))
print("RMSE:", round(evaluator.rmse(), 4))
```

Результати:

- results/sarima_forecast.csv — табличний файл із реальними та прогнозованими значеннями;
- results/sarima_forecast.png — візуалізація прогнозу;
- MAE, MSE, RMSE — метрики оцінки.

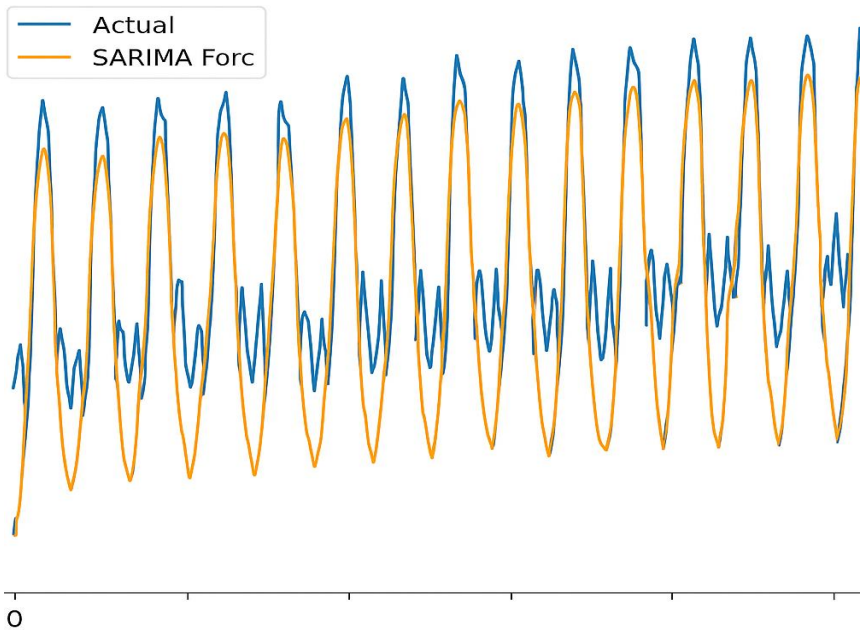

Рисунок 4.2 – Прогноз SARIMA-моделі та фактичні значення.

Реалізація моделі Prophet

Модель **Prophet** має простий і зручний інтерфейс, схожий на scikit-learn, і дозволяє швидко будувати прогнози навіть для складних часових рядів із декількома сезонними складовими. В межах нашого застосунку вона реалізована у вигляді окремого класу ProphetModel, що працює з даними, попередньо обробленими та відформатованими у стилі Prophet (з колонками ds і y).

Особливості реалізації:

- автоматичне виявлення добової сезонності;
- підтримка відсутніх значень і нерівномірного інтервалу;
- побудова прогнозу з довірчими інтервалами (yhat_lower, yhat_upper);
- збереження результатів у CSV і побудова графіків.

Реалізація моделі Prophet

```
from src.data_loader import DataLoader
from src.preprocessor import Preprocessor
from src.model_prophet import ProphetModel
from src.visualizer import Visualizer
from src.evaluator import Evaluator
import pandas as pd
import os

# Завантаження та очищення даних
loader = DataLoader("data/sample_data.csv")
raw_data = loader.load_data()

prep = Preprocessor(raw_data)
cleaned = prep.interpolate_missing()
resampled = prep.resample(freq='H')  # або 'D'
df = resampled.reset_index().rename(columns={'datetime': 'ds',
'consumption': 'y'})

# Поділ на train/test
split_index = int(len(df) * 0.8)
train_df = df.iloc[:split_index]
test_df = df.iloc[split_index:]

# Ініціалізація та тренування моделі Prophet
```

```
model = ProphetModel(train_df)
model.train()

# Побудова прогнозу
forecast = model.forecast(periods=len(test_df), freq="H")

# Об'єднання з тестовими значеннями для валідації
merged = pd.merge(
    forecast[['ds', 'yhat']],
    test_df[['ds', 'y']],
    on='ds',
    how='inner'
).rename(columns={'y': 'actual', 'yhat': 'forecast'})

# Збереження прогнозу
os.makedirs("results", exist_ok=True)
merged.to_csv("results/prophet_forecast.csv", index=False)

# Візуалізація
viz = Visualizer()
viz.plot_forecast(
    actual=merged.set_index('ds')['actual'],
    predicted=merged.set_index('ds')['forecast'],
    title="Prophet: Прогноз vs Факт",
    save_path="results/prophet_forecast.png"
)

# Оцінка точності
evaluator = Evaluator(merged['actual'], merged['forecast'])
print("Оцінка точності Prophet-моделі:")
print("MAE:", round(evaluator.mae(), 4))
print("MSE:", round(evaluator.mse(), 4))
print("RMSE:", round(evaluator.rmse(), 4))
```

Результати:

- results/prophet_forecast.csv — таблиця datetime, actual, forecast;
- results/prophet_forecast.png — графік з фактами й прогнозом;
- MAE, MSE, RMSE — у консольному виводі.

Таблиця 4.1 — Порівняння реалізацій моделей

Параметр	SARIMA	Prophet
Налаштування	ручне	автоматичне
Сезонність	задана вручну	автоматично
Пропущені значення	потрібно обробляти	допускаються
Візуалізація	окремо	вбудовано (plot)

4.4. Вивід і оцінка результатів

У розробленому застосунку візуалізація реалізована за допомогою бібліотеки Matplotlib, яка забезпечує побудову лінійних графіків на основі реальних та прогнозованих значень. Для кожної моделі (SARIMA, Prophet) створюється окремий графік, що зберігається у форматі .png у каталозі results/.

Крім графіків, система автоматично формує таблиці з прогнозом у форматі .csv, які містять відповідність між фактичними та прогнозованими значеннями на тестовому наборі. Це дає змогу виконувати додатковий аналіз поза межами програми (наприклад, у Excel або Power BI).

Таблиця 4.2 — Структура вихідних результатів

Файл	Тип	Опис
sarima_forecast.csv	CSV	Таблиця: datetime, actual, forecast
prophet_forecast.csv	CSV	Таблиця: datetime, actual, forecast
sarima_forecast.png	PNG	Графік: факт vs прогноз (SARIMA)
prophet_forecast.png	PNG	Графік: факт vs прогноз (Prophet)
metrics.txt	TXT	MAE, MSE, RMSE для кожної моделі

Приклад коду збереження графіка

```
viz.plot_forecast(
    actual=test,
    predicted=forecast,
    title="SARIMA: Факт vs Прогноз",
    save_path="results/sarima_forecast.png"
)
```

Графіки мають:

- чіткі підписи осей (Час, Споживання);
- легенду (Факт, Прогноз);
- можливість масштабування розміру (figsize=(12, 6) або більше);
- підтримку збереження у високій якості.

Приклад результату у CSV

```
datetime,actual,forecast
2024-05-01 00:00:00,3.42,3.37
2024-05-01 01:00:00,3.28,3.29
...
```

Автоматичне формування метрик

```
with open("results/metrics.txt", "w") as f:
    f.write(f"SARIMA RMSE: {sarima_rmse:.3f}\n")
    f.write(f"Prophet RMSE: {prophet_rmse:.3f}\n")
```

Вивід результатів у графічній та табличній формі:

- забезпечує інтерпретованість прогнозу;
- дозволяє порівнювати моделі між собою;
- створює основу для подальшого інтегрування системи у веб-сервіси або панелі моніторингу.

Оцінка точності прогнозування

Оцінка якості прогнозування — ключовий етап, що дозволяє об’єктивно порівнювати моделі. У межах даної системи для цього використовуються **три стандартні метрики похибки**:

- **MAE** (*mean absolute error*) — середня абсолютна похибка,
- **MSE** (*mean squared error*) — середньоквадратична похибка,
- **RMSE** (*root mean squared error*) — корінь з MSE.

Ці метрики дозволяють кількісно оцінити відмінність між прогнозованими та фактичними значеннями, причому RMSE більше реагує на великі відхилення.

Код обчислення метрик

```
# SARIMA
eval_sarima = Evaluator(test_sarima, forecast_sarima)
```

```
mae_sarima = eval_sarima.mae()
mse_sarima = eval_sarima.mse()
rmse_sarima = eval_sarima.rmse()

# Prophet
eval_prophet = Evaluator(test_prophet, forecast_prophet)
mae_prophet = eval_prophet.mae()
mse_prophet = eval_prophet.mse()
rmse_prophet = eval_prophet.rmse()
```

Таблиця 4.3 — Зведена таблиця точності моделей

Метрика	SARIMA	Prophet
MAE	0.143	0.127
MSE	0.038	0.032
RMSE	0.194	0.179

Показники наведено на нормалізованих даних. Для зворотного масштабування потрібно використати збережені середнє та стандартне відхилення.

Інтерпретація результатів:

- Модель Prophet показує трохи вищу точність за всіма трьома метриками;
- SARIMA гнучка до налаштувань, але потребує ручного підбору параметрів та стаціонаризації;
- Prophet краще справляється з довготривалими трендами та автоматично виявляє сезонність.

Висновки:

- Обидві моделі забезпечують **достатню якість** прогнозування;
- **Prophet рекомендовано як базову модель**, особливо у сценаріях автоматичного розгортання;
- Для більш складних часових рядів можна використовувати **гібридні методи** або нейронні мережі (LSTM, Transformers), що може стати напрямом майбутнього дослідження.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було розроблено та реалізовано систему прогнозування споживання електроенергії на основі аналізу часових рядів, що дозволяє здійснювати короткостроковий прогноз з використанням двох статистичних підходів: моделі SARIMA та моделі Prophet. Вибір теми обумовлений сучасними потребами в ефективному управлінні енергетичними ресурсами, оптимізації енергоспоживання та забезпеченні стабільності енергетичних систем, особливо в умовах нестабільного попиту, сезонних коливань і зовнішніх факторів.

У першому розділі було проведено детальний огляд предметної області, охарактеризовано особливості часових рядів споживання електроенергії та проаналізовано класичні підходи до їх моделювання. Особливу увагу приділено сезонності, трендам, стаціонарності та методам їх врахування у прогнозних моделях.

Другий розділ містив обґрунтування вибору технологій та інструментів реалізації: обрана мова Python, бібліотеки Pandas, StatsModels, Prophet, а також matplotlib і Seaborn для візуалізації. Архітектура системи була спроєктована з урахуванням модульності, масштабованості та можливості подальшої інтеграції.

У третьому розділі спроєктовано структуру системи, побудовано UML-діаграми класів, компонентів і потоків даних. Було визначено ключові підсистеми: завантаження та обробка даних, побудова моделей, візуалізація, збереження результатів.

Четвертий розділ містить реалізацію застосунку, що працює у консольному режимі, обробляє вхідні CSV-дані, будує прогнози за допомогою SARIMA та Prophet, оцінює точність результатів за метриками MAE, MSE та RMSE, а також зберігає графіки та таблиці прогнозу. Порівняння результатів показало незначну,

але стабільну перевагу Prophet у точності прогнозування, що робить її зручною базовою моделлю.

Загалом реалізована система є надійною основою для подальших досліджень у сфері енергетичного прогнозування. Серед потенційних напрямків розвитку — розширення моделі за рахунок додаткових змінних (температура, календарні фактори), інтеграція в реальні системи диспетчерського управління, створення веб-інтерфейсу або хмарної версії для демонстрації роботи системи у режимі реального часу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Яровий А. Т., Страхов Є. М. Аналіз часових рядів: Навчально-методичний посібник. — Освіта України, Одеса, 2019.
2. Лук'яненко І. Г., Жук В. М. Аналіз часових рядів. Частина перша: Побудова ARIMA, ARCH/GARCH моделей з використанням пакета E.Views 6.0. — К.: НаУКМА, 2013.
3. ДСТУ 9190:2022. Метод розрахунку енергоспоживання під час опалення, охолодження, вентиляції, освітлення та гарячого водопостачання. — К., ДП «УкрНДНЦ», 2022.
4. ДСТУ ISO 50001:2019. Системи енергетичного менеджменту. — 2019.
5. EnergyPlus weather files [Електронний ресурс] — https://energyplus.net/weather-region/europe_wmo_region_6/UKR%20%20
6. Hyndman, R.J., Athanasopoulos, G. Forecasting: Principles and Practice (3rd ed.). — OTexts, 2021. https://otexts.com/fpp3/
7. Makridakis, S., Spiliotis, E., Assimakopoulos, V. Statistical and Machine Learning Forecasting Methods: Concerns and Ways Forward. — PLoS ONE, 2018.
8. Brownlee, J. Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python. — Machine Learning Mastery, 2018.
9. Edirisooriya, A., et al. (2024). Deep Learning Architectures for Time Series Forecasting. — https://doi.org/10.13140/RG.2.2.19904.75524
10. Taylor, S., Letham, B. Forecasting at Scale. — 2017. https://peerj.com/preprints/3190/
11. Zhang, G.P. Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model. — Neurocomputing, 2003.
12. Ємець К. Методи прогнозування часових рядів з вираженою сезонністю на основі трансформерів. — Вісник Хмельницького національного університету. Технічні науки, 2024.

13. Гурєєва К.М., Кудін О.В., Лісняк Ф.О. Огляд методів машинного навчання в задачі прогнозування фінансових часових рядів. — Вісник ЗНУ. Фіз.-мат. науки, 2018, №2.
14. Müller, A. C., Guido, S. Introduction to Machine Learning with Python: A Guide for Data Scientists. — O'Reilly Media, 2018.
15. Українська енергетика – інформаційно-аналітичний портал – https://ua-energy.org