Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: "Розробка системи скрапінгу онлайн-магазинів одягу з функціями аналізу та прогнозу"

Виконав(ла): студент(ка) VI курсу, групи СНнм-61
спеціальності 122 "Комп'ютерні науки"
(шифр і назва спеціальності)

	(підпис)	Караванський В.В. (прізвище та ініціали)
Керівник	(підпис)	Боднарчук І.О. (прізвище та ініціали)
Нормоконтроль	(підпис)	(прізвище та ініціали)
Завідувач кафедри	(підпис)	Боднарчук І.О. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп’ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
"__" _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)

Студенту Караванському Владиславу Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка системи скрапінгу онлайн-магазинів одягу з функціями аналізу та прогнозу

Керівник роботи канд. техн. наук, доцент Боднарчук Ігор Орестович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від "10" березня 2026 року № 4/9-150

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Набір даних зібраний з онлайн-магазину, проаналізовані дані і висновок/прогноз програми щодо них.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ, 1.1 Огляд предметної області, 1.2 Етичні та правові аспекти використання веб-скрапінгу у маркетингових дослідженнях, 1.2.1 Теоретичні основи веб-скрапінгу в маркетингових дослідженнях, 1.2.2 Правові аспекти використання веб-скрапінгу, 1.2.3 Етичні принципи та рекомендації щодо використання веб-скрапінгу, 1.3 Аналіз методів визначення популярних товарних підкатегорій та прогнозування їх популярності, 1.4 Актуальність обраного дослідження, 2.1 Обґрунтування вибору мови програмування Python, 2.2 Застосовані об'єктно-орієнтовані технології та принципи, 2.3 Логіка аналізу отриманих результатів, 3.1 Збір даних (Scraper), 3.2 Аналіз часових рядів і прогнозування, 4.1 Охорона праці, 4.1.1 Аналіз небезпечних та шкідливих факторів під час розробки й експлуатації системи скрапінгу онлайн-магазинів одягу, 4.1.2 Організація безпечного робочого місця програміста при розробці системи аналізу та прогнозування даних онлайн-магазинів, 4.1.3 Висновок до розділу "Охорона праці", 4.2 Безпека в надзвичайних ситуаціях, 4.2.1 Забезпечення безпеки роботи з комп’ютерною технікою в умовах надзвичайних ситуацій, 4.2.2 Висновок до розділу "Безпека в надзвичайних ситуаціях", Висновки, Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
Рисунки: приклади візуалізації результатів, графіки популярності пошукових запитів, скріншоти з результатами виконання завдання, скріншоти з вмістом конференцій.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	кандидат технічних наук, доцент кафедри		
	МТ Гурик Олег Ярославович		
Безпека в надзвичайних	проректор з адміністративно-господарської		
ситуаціях	роботи та будівництва Теслюк Володимир		
	Миколайович		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	13.04.2026	*Виконано*
2.	Підбір та опрацювання кваліфікаційних публікацій,	13.04.2026-20.04.2026	*Виконано*
	збір даних по темі роботи		
3.	Виконання дослідження згідно теми кваліфікаційної	21.04.2026-03.05.2026	*Виконано*
	роботи		
4.	Оформлення розділу "Аналіз предметної області"	04.05.2026-10.05.2026	*Виконано*
5.	Оформлення розділу "Аналіз методів, систем та	04.05.2026-10.05.2026	*Виконано*
	проєктування"		
6.	Оформлення розділу "Демонстрація, аналіз та	04.05.2026-10.05.2026	*Виконано*
	узагальнення отриманих результатів"		
7.	Виконання завдання до підрозділу "Охорона праці"	27.04.2026-10.05.2026	*Виконано*
8.	Виконання завдання до підрозділу "Безпека в	27.04.2026-10.05.2026	*Виконано*
	надзвичайних ситуаціях"		
9.	Оформлення розділу "Охорона праці та безпека в	27.04.2026-10.05.2026	*Виконано*
	надзвичайних ситуаціях"		
10.	Оформлення кваліфікаційної роботи	11.05.2026-13.05.2026	*Виконано*
11.	Нормоконтроль	14.05.2026	*Виконано*
12.	Перевірка на плагіат	15.05.2026	*Виконано*
13.	Попередній захист кваліфікаційної роботи	18.05.2026	*Виконано*
14.	Захист кваліфікаційної роботи	26.05.2026	

Студент _____ Караванський В. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка системи скрапінгу онлайн-магазинів одягу з функціями аналізу та прогнозу // Кваліфікаційна робота освітнього ступеня "Магістр" // Караванський Владислав Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНнм-61 // Тернопіль, 2026 // С. 146, рис. – 32, табл. – 4, додат. – 2, бібліогр. – 25.

Ключові слова: скрапінг, веб-сайт, аналіз, дані, онлайн-магазин, прогноз, python.

Кваліфікаційна робота присв'ячена розробці системи скрапінгу онлайн-магазинів одягу з функціями аналізу та прогнозу.

В першому розділі кваліфікаційної роботи описані етичні та правові аспекти використання веб-скрапінгу у маркетингових дослідженнях. Здійснено аналіз методів визначення популярних товарних підкатегорій та прогнозування їх популярності. Підтверджено актуальність обраного дослідження.

В другому розділі кваліфікаційної роботи описано аналіз методів, систем та проєктування. Обґрунтовано вибір мови програмування Python. Описано застосовані об'єктно-орієнтовані технології та принципи. Досліджено логіку аналізу отриманих результатів.

В третьому розділі кваліфікаційної роботи здійснено демонстрацію, аналіз та узагальнення отриманих результатів. Продемонстровано збір даних. Проведено аналіз часових рядів і прогнозування. Об'єкт дослідження: процеси збору, аналізу та прогнозування даних онлайн-магазинів одягу. Предмет дослідження: методи та програмні засоби веб-скрапінгу, аналізу часових рядів і прогнозування популярності товарних підкатегорій онлайн-магазинів одягу.

ANNOTATION

Development of a Web Scraping System for Online Clothing Stores with Analysis and Forecasting Functions // The educational level "Master" qualification work // Vladyslav Karavanskyy // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNnm-61 group // Ternopil, 2026 // P. 146, fig. – 32, tables – 4, annexes – 2, ref. – 25.

Keywords: scraping, website, analysis, data, online store, forecast, python.

The qualification work is dedicated to the development of a scraping system for online clothing stores with analysis and forecast functions.

The first section of the qualification work describes the ethical and legal aspects of using web scraping in marketing research. The analysis of methods for determining popular product subcategories and predicting their popularity is carried out. The relevance of the selected study is confirmed.

The second section of the qualification work describes the analysis of methods, systems and design. The choice of the Python programming language is justified. The applied object-oriented technologies and principles are described. The logic of analyzing the obtained results is investigated.

The third section of the qualification work demonstrates, analyzes and summarizes the obtained results. Data collection is demonstrated. Time series analysis and forecasting are carried out. The object of research: the processes of collecting, analyzing and predicting data for online clothing stores. Subject of research: methods and software tools for web scraping, time series analysis, and forecasting the popularity of product subcategories of online clothing stores.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Абстракція (англ. – abstraction) – принцип ООП, що передбачає виділення суттєвих характеристик об'єкта та приховування несуттєвих деталей реалізації.

Асинхронність (англ. – asynchrony) – модель виконання програми, за якої операції виконуються незалежно одна від одної, не блокуючи основний потік виконання.

БД – база даних.

Веб-сканування (англ. – web crawling) – автоматичний обхід веб-сторінок за посиланнями з метою їх індексації або збору даних.

Веб-скрапінг (англ. – web scraping) – автоматизований збір даних із веб-сторінок за допомогою спеціального програмного забезпечення.

Декомпозиція часового ряду – розкладання ряду на складові: тренд, сезонність та залишок.

Диференціювання (англ. – differencing) – перетворення часового ряду шляхом обчислення різниць між послідовними значеннями для досягнення стаціонарності.

Довірчий інтервал (англ. – confidence interval) – діапазон значень, у якому з заданою імовірністю знаходиться прогнозоване значення.

Інкапсуляція (англ. – encapsulation) – принцип ООП, що полягає у приховуванні внутрішньої реалізації об'єкта та наданні доступу до нього лише через визначений інтерфейс.

Машинне навчання (англ. – machine learning) – розділ штучного інтелекту, що вивчає методи побудови алгоритмів, які навчаються на основі даних.

ООП – об'єктно-орієнтоване програмування.

Парсинг (англ. – parsing) – синтаксичний аналіз структури документа (наприклад, HTML-сторінки) з метою вилучення необхідних даних.

Поліморфізм (англ. – polymorphism) – принцип ООП, що дозволяє об'єктам різних класів оброблятися через єдиний інтерфейс батьківського класу.

Сезонність (англ. – seasonality) – регулярні циклічні коливання у часовому ряді, що повторюються через фіксовані проміжки часу.

Стаціонарність (англ. – stationarity) – властивість часового ряду, за якої його середнє значення, дисперсія та автокореляційна структура не змінюються з часом.

Тренд (англ. – trend) – довгострокова спрямованість зміни показника часового ряду.

Успадкування (англ. – inheritance) – принцип ООП, що дозволяє новому класу отримувати властивості та методи вже існуючого класу.

Часовий ряд (англ. – time series) – впорядкована в часі послідовність спостережень певного показника.

ACF (Autocorrelation Function) – функція автокореляції.

ADF (Augmented Dickey-Fuller test) – розширений тест Дікі-Фуллера на наявність одиничного кореня.

AIC (Akaike Information Criterion) – інформаційний критерій Акайке.

API (Application Programming Interface) – інтерфейс прикладного програмування.

ARIMA (AutoRegressive Integrated Moving Average) – авторегресійна інтегрована ковзна середня.

CSV (Comma-Separated Values) – текстовий формат збереження табличних даних із роздільником-комою.

DIP (Dependency Inversion Principle) – принцип інверсії залежностей.

GDPR (General Data Protection Regulation) – Загальний регламент про захист персональних даних Європейського Союзу.

HTML (HyperText Markup Language) – мова гіпертекстової розмітки.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

ISP (Interface Segregation Principle) – принцип розділення інтерфейсів.

JSON (JavaScript Object Notation) – текстовий формат обміну даними на основі JavaScript.

KPSS (Kwiatkowski-Phillips-Schmidt-Shin test) – тест Квятковського-Філліпса-Шмідта-Шина на стаціонарність часового ряду.

LSP (Liskov Substitution Principle) – принцип заміщення Барбари Ліскова.

MA (Moving Average) – ковзна середня.

OCP (Open-Closed Principle) – принцип відкритості/закритості.

PACF (Partial Autocorrelation Function) – функція часткової автокореляції.

SARIMA (Seasonal AutoRegressive Integrated Moving Average) – сезонна авторегресійна інтегрована модель ковзної середньої.

SOLID – абревіатура п'яти основних принципів об'єктно-орієнтованого проєктування: SRP, OCP, LSP, ISP, DIP.

SRP (Single Responsibility Principle) – принцип єдиної відповідальності.

STL (Seasonal and Trend decomposition using Loess) – метод декомпозиції часового ряду на трендову, сезонну та залишкову складові.

ToS (Terms of Service) – умови надання послуг (користувацька угода веб-сайту).

URL (Uniform Resource Locator) – уніфікований локатор ресурсів; адреса веб-сторінки.

ЗМІСТ

ВСТУП

Актуальність теми. Стрімкий розвиток електронної комерції призводить до постійного зростання обсягів інформації, що генерується онлайн-магазинами. Особливо динамічно розвивається сегмент продажу одягу, де асортимент товарів, ціни, категорії та підкатегорії продукції регулярно оновлюються відповідно до змін споживчого попиту та модних тенденцій. У таких умовах аналіз великих обсягів даних стає важливим інструментом для виявлення закономірностей ринку, визначення популярних товарних категорій та прогнозування майбутніх трендів. Значна частина інформації про товари, їх характеристики та структуру каталогів знаходиться у веб-ресурсах інтернет-магазинів, однак ці дані зазвичай представлені у вигляді веб-сторінок, що ускладнює їх безпосереднє використання для аналітичних досліджень.

Технології веб-скрапінгу дозволяють автоматизувати процес збору даних з веб-ресурсів, забезпечуючи отримання великого обсягу структурованої інформації за короткий проміжок часу. Проте сучасні комерційні рішення для збору та аналізу даних з інтернет-магазинів часто є закритими системами або пропонують обмежений функціонал для дослідження структури ринку. Крім того, недостатньо дослідженими залишаються питання автоматизованого визначення популярних підкатегорій товарів на основі даних електронної комерції та прогнозування їх популярності у майбутньому із застосуванням методів аналізу даних і машинного навчання.

У зв’язку з цим розроблення просунутої системи веб-скрапінгу онлайн-магазинів з продажу одягу, яка забезпечує автоматизований збір, збереження та аналітичне опрацювання даних, а також дозволяє визначати популярні підкатегорії одягу та прогнозувати їх популярність у майбутньому, є актуальним напрямком сучасних наукових досліджень у сфері інформаційних технологій та аналізу даних.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня "Магістр" є розроблення просунутої системи веб-скрапінгу онлайн-магазинів з продажу одягу з можливістю виявлення популярних підкатегорій одягу та прогнозування їх популярності у майбутньому.

Для досягнення поставленої мети необхідно виконати ряд завдань, зокрема:

– проаналізувати сучасний стан досліджень у сфері веб-скрапінгу та аналізу даних електронної комерції;

– дослідити існуючі методи збору даних з веб-ресурсів та інтернет-магазинів;

– проаналізувати методи виявлення популярних товарних категорій на основі зібраних даних;

– дослідити підходи до прогнозування популярності товарних підкатегорій із використанням методів аналізу часових рядів;

– виконати порівняльний аналіз існуючих інструментів та підходів до збору й аналізу даних з онлайн-магазинів;

– розробити систему автоматизованого збору даних з онлайн-магазинів одягу;

– реалізувати модуль аналізу та визначення популярних підкатегорій одягу;

– розробити модуль прогнозування популярності підкатегорій одягу на основі зібраних даних.

Об’єкт дослідження – процеси збору та аналітичного опрацювання даних онлайн-магазинів з продажу одягу.

Предмет дослідження – методи веб-скрапінгу, аналізу та прогнозування популярності підкатегорій одягу на основі даних електронної комерції.

Наукова новизна одержаних результатів кваліфікаційної роботи полягає у тому, що отримав подальший розвиток підхід до автоматизованого аналізу структури онлайн-магазинів одягу на основі веб-скрапінгу з можливістю визначення популярних підкатегорій товарів та прогнозування їх популярності із застосуванням методів аналізу даних.

Практичне значення одержаних результатів. Розроблено прототип програмної системи веб-скрапінгу онлайн-магазинів одягу, який забезпечує автоматизований збір даних про товари, аналіз структури категорій та підкатегорій, а також визначення та прогнозування популярності підкатегорій одягу.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя.

Публікації. Основні результати кваліфікаційної роботи опубліковано у матеріалах наукової конференції (див. додатки А).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку джерел з 25 найменувань та 2 додатків. Загальний обсяг кваліфікаційної роботи складає 146 сторінок, з них 58 сторінок основного тексту, який містить 32 рисунки, 4 таблиці та 11 формул.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Для переважної більшості користувачів комп'ютерів взаємодія з Інтернетом відбувається у формі доступу до веб-сайтів через браузер, де інформація та мультимедійні об'єкти відображаються у зручному для людини вигляді для легкого розуміння. Однак це лише верхівка айсберга щодо потенціалу Інтернету, оскільки існує так багато корисної необробленої інформації, яка прихована від очей. API можна використовувати на багатьох сайтах для легкого доступу до більшої частини цієї інформації, але це залежить від власника сайту, і якщо він вирішить не робити цю інформацію доступною через API, цього легко не зробити. З іншого боку, веб-скрапінг або веб-сканування набагато швидше та ефективніше, і його можна використовувати для збору та компіляції даних з тисяч або мільйонів сторінок для обробки та отримання інформації. Цей метод є безцінним у широкому спектрі застосувань, але зокрема в галузі бізнес-аналітики [1].

Цінність веб-скрапінгу важко заперечити, якщо ти бізнесмен, який прагне отримати якнайбільше інформації про конкурентів, чи потенційних партнерів – ти повинен бути оснащений інформацією. В епоху, коли дані правлять світо – це чи не найважливіша "зброя", якою прагнуть заволодіти всі. Якщо це викликає сумніви, то варто лише згадати лист Марка Цукерберга до працівників своєї компанії.

9 червня 2016 року генеральний директор Facebook Марк Цукерберг поскаржився на відсутність аналітики щодо конкурента Snapchat: "Щоразу, коли хтось ставить питання про Snapchat, відповідь зазвичай така: оскільки їхній трафік зашифрований, у нас немає аналітики про них... Враховуючи, як швидко вони зростають, важливо знайти новий спосіб отримання надійної аналітики про них. Можливо, нам потрібно створювати панелі або писати спеціальне програмне забезпечення. Вам слід з'ясувати, як це зробити." [2]. Варто зазначити, що в результаті команда використала не зовсім легальний спосіб отримання цих даних,

щоб призвело до розляду цього випадку у суді [2]. Звісно, не обговорюється, що етичність веб-скрапінгу – один з найважливіших аспектів цього роду діяльності і завжди потрібно робити речі в рамках закону. Але цей приклад показує, що дані – важливі і без них не обійтися в сучасному світі інформаційниї технологій.

Сучасні компанії працюють у середовищі надзвичайно швидких змін, де конкурентні переваги часто визначаються швидкістю отримання та обробки інформації. Дані про ціни, попит, поведінку користувачів, нові продукти або маркетингові стратегії конкурентів дозволяють бізнесу приймати обґрунтовані рішення. Веб-скрапінг, у цьому контексті, виступає одним із ключових інструментів збору таких даних. На відміну від традиційних методів дослідження ринку, які можуть вимагати значних фінансових та часових ресурсів, автоматизоване збирання інформації з веб-сайтів дозволяє отримувати актуальні дані у великому масштабі та з високою швидкістю.

Однією з найбільш поширених сфер використання веб-скрапінгу є електронна комерція. Інтернет-магазини регулярно відстежують ціни конкурентів, асортимент товарів, наявність акцій та знижок. Завдяки цьому вони можуть оперативно коригувати власну цінову політику, пропонуючи більш конкурентні умови для покупців. Крім того, аналіз відгуків користувачів, рейтингів товарів і популярності певних категорій дозволяє компаніям краще розуміти потреби клієнтів і адаптувати свою продукцію або послуги відповідно до реального попиту.

Ще одним важливим напрямком застосування веб-скрапінгу є аналітика ринку праці. Дані з сайтів вакансій можуть використовуватися для аналізу попиту на певні професії, визначення середнього рівня заробітної плати або дослідження технологічних трендів. Наприклад, аналіз великої кількості вакансій дозволяє виявити, які мови програмування чи інструменти стають більш популярними у галузі. Такі дослідження корисні не лише для компаній, але й для освітніх установ, які прагнуть адаптувати навчальні програми до реальних потреб ринку.

У сфері фінансів веб-скрапінг також відіграє важливу роль. Інвестори та аналітики використовують його для збору новин, фінансових звітів, котирувань

акцій або інформації про діяльність компаній. Обробка цих даних дозволяє створювати моделі прогнозування, виявляти ринкові тенденції та приймати більш обґрунтовані інвестиційні рішення. У багатьох випадках швидкість отримання інформації може прямо впливати на фінансовий результат, оскільки навіть невелика затримка може призвести до втрати можливостей на ринку.

Веб-скрапінг також активно використовується у наукових дослідженнях. Наприклад, соціологи можуть аналізувати публікації у соціальних мережах для дослідження суспільних настроїв, а економісти – відстежувати зміну цін на товари або послуги. У сфері журналістики дані, отримані за допомогою автоматизованого збору інформації, часто стають основою для аналітичних матеріалів та розслідувань.

Однак, поряд із численними перевагами, веб-скрапінг має і низку викликів. Перш за все, це питання етики та законності. Багато веб-сайтів мають власні правила використання, які можуть обмежувати автоматизований доступ до їхнього контенту. Крім того, необхідно враховувати питання конфіденційності та захисту персональних даних. Саме тому відповідальне використання веб-скрапінгу передбачає дотримання законодавства, повагу до правил веб-ресурсів та використання отриманих даних лише у легальний спосіб.

Ще одним викликом є технічна складність сучасних веб-сайтів. Багато ресурсів використовують динамічне завантаження контенту, складні механізми автентифікації або системи захисту від автоматизованих запитів. Це означає, що для ефективного збору даних розробникам доводиться застосовувати складніші інструменти та підходи, включаючи роботу з API, емуляцію поведінки користувача або використання спеціалізованих бібліотек для взаємодії з динамічними сторінками.

Веб-скрапінг включає різноманітні методики збору даних із веб-ресурсів. Вибір конкретного підходу залежить від структури сайту, типу даних, що необхідно отримати, а також від технічних обмежень веб-ресурсу. Найпопулярніші методики веб-скрапінгу можна умовно поділити на декілька категорій.

Парсинг HTML-сторінок. Однією з найпоширеніших методик є безпосередній аналіз HTML-коду веб-сторінки. У цьому випадку програма надсилає HTTP-запит до веб-сервера, отримує HTML-документ і виділяє з нього необхідні елементи за допомогою структури DOM. Для цього зазвичай використовуються спеціальні бібліотеки або інструменти, які дозволяють знаходити елементи за тегами, класами, ідентифікаторами або іншими атрибутами. Такий підхід добре працює для сайтів зі статичним контентом, де всі необхідні дані вже присутні у вихідному HTML-коді сторінки.

HTML-парсинг широко застосовується для:

– збору інформації про товари в інтернет-магазинах;

– аналізу новинних сайтів;

– отримання списків компаній, контактів або інших структурованих даних.

Перевагою цього методу є його відносна простота та висока швидкість виконання. Однак він може бути менш ефективним для сучасних веб-сайтів, які активно використовують динамічне завантаження контенту.

Використання API веб-сайтів. Багато сучасних веб-сервісів надають офіційні або неофіційні API (Application Programming Interface), через які можна отримувати структуровані дані у форматах JSON або XML (RSS-фіди). У такому випадку скрапінг полягає у відправленні запитів безпосередньо до API-сервера та отриманні даних у зручному для обробки вигляді. Цей підхід значно спрощує процес обробки інформації, оскільки немає необхідності аналізувати HTML-код.

Метод використання API застосовується для:

– отримання даних із соціальних мереж;

– збору статистики та аналітичної інформації;

– інтеграції даних із веб-сервісів у власні системи.

Основною перевагою є стабільність та структурованість даних. Проте API часто мають обмеження щодо кількості запитів або потребують автентифікації.

Скрапінг динамічних веб-сторінок. Багато сучасних сайтів використовують технології JavaScript для динамічного завантаження контенту після початкового

завантаження сторінки. У таких випадках звичайний HTTP-запит не дозволяє отримати всі необхідні дані. Для вирішення цієї проблеми застосовується емуляція роботи браузера. Спеціальні інструменти запускають справжній браузер або його безголову (headless) версію, виконують JavaScript-код сторінки та лише після цього отримують повністю сформований HTML-документ.

Цей підхід використовується для:

– роботи з веб-додатками з інтенсивним використанням JavaScript;
– збору даних із платформ електронної комерції;
– отримання інформації із соціальних мереж.

Недоліком цього методу є значно більші витрати ресурсів і часу на обробку.

1.2 Етичні та правові аспекти використання веб-скрапінгу у маркетингових дослідженнях

1.2.1 Теоретичні основи веб-скрапінгу в маркетингових дослідженнях

Припустимо, вам потрібна певна інформація з веб-сайту. Що ви робите? Ну, ви можете скопіювати та вставити інформацію з Вікіпедії у свій файл. Але що, якщо ви хочете отримати великі обсяги інформації з веб-сайту якомога швидше? Наприклад, великі обсяги даних з веб-сайту для навчання алгоритму машинного навчання? У такій ситуації копіювання та вставка не працюватиме. І саме тоді вам знадобиться веб-скрапінг. На відміну від тривалого та виснажливого процесу ручного отримання даних, веб-скрапінг використовує методи автоматизації, щоб отримати тисячі або навіть мільйони наборів даних за менший проміжок часу [3].

Існує багато різних способів виконання веб-скрапінгу для отримання даних з веб-сайтів. До них належать використання онлайн-сервісів, певних API або навіть створення власного коду для веб-скрапінгу з нуля. Багато великих веб-сайтів, таких як Google, Twitter, Facebook, StackOverflow тощо, мають API, які дозволяють вам отримувати доступ до їхніх даних у структурованому форматі. Це найкращий варіант, але є й інші сайти, які не дозволяють користувачам

отримувати доступ до великих обсягів даних у структурованій формі або вони просто не настільки технологічно розвинені. У такій ситуації найкраще використовувати веб-скрапінг для збору даних з веб-сайту [3].

Веб-скрапінг вимагає двох частин, а саме: сканера та скрапера. Сканер – це алгоритм штучного інтелекту, який переглядає веб-сторінки, щоб знайти потрібні дані, переходячи за посиланнями в інтернеті. Скрапер, з іншого боку, – це спеціальний інструмент, створений для вилучення даних з веб-сайту. Дизайн скрапера може значно відрізнятися залежно від складності та обсягу проекту, щоб він міг швидко та точно вилучати дані [3].

Веб-скрапери можуть витягувати всі дані на певних сайтах або конкретні дані, які потрібні користувачеві. В ідеалі, найкраще вказати потрібні дані, щоб веб-скрапер швидко витягував лише ці дані. Наприклад, ви можете захотіти витягнути зі сторінки Amazon типи доступних соковижималок, але вам можуть знадобитися лише дані про моделі різних соковижималок, а не відгуки клієнтів [3].

Отже, коли веб-скраперу потрібно витягнути з сайту URL-адреси. Потім він завантажує весь HTML-код для цих сайтів, а більш просунутий скрапер може навіть витягти всі елементи CSS та Javascript. Потім скрапер отримує необхідні дані з цього HTML-коду та виводить їх у форматі, вказаному користувачем. Здебільшого це у формі електронної таблиці Excel або файлу CSV, але дані також можна зберігати в інших форматах, таких як файл JSON [3].

1.2.2 Правові аспекти використання веб-скрапінгу

Веб-скрапінг не є прямо незаконним. Немає конкретних законів, які забороняють веб-скрапінг. Багато компаній використовують його легально для збору цінних даних за допомогою різних інструментів веб-скрапінгу [4].

Однак певні ситуації можуть зробити веб-скрапінг незаконним [4].

Порушення умов надання послуг: Вхід на веб-сайти та скрапінг даних може бути проблемою. Коли ви входите в систему, ви погоджуєтеся з Умовами надання послуг (ToS) веб-сайту, які часто забороняють автоматизований збір даних [4].

Помилка щодо публічних даних: Загальнодоступні дані не завжди можна використовувати без обмежень. Навіть із публічними даними ви повинні бути обережними, щоб не порушувати закони, особливо щодо авторського права [4].

Творча робота: Завантаження матеріалів, захищених авторським правом, таких як статті, відео чи дизайни, зазвичай є незаконним. Цей матеріал захищений законами про авторське право [4].

Автоматичний збір даних: Деякі Умови надання послуг можуть забороняти будь-який автоматичний збір даних, незалежно від цільового використання даних [4]. У цих випадках сама діяльність зі скрапінгу може бути незаконною, а не лише використання даних [4].

Закони про конфіденційність суттєво впливають на веб-скрапінг, особливо на роботу з персональними даними [4].

Загальний регламент про захист даних (GDPR) – це закон Європейського Союзу про конфіденційність, який набрав чинності 25 травня 2018 року. Він спрямований на те, щоб надати громадянам ЄС контроль над своєю особистою інформацією. GDPR не робить веб-скрапінг незаконним, але обмежує те, як підприємства використовують зібрані дані. Наприклад, підприємствам часто потрібна чітка згода фізичних осіб на збір та використання їхніх персональних даних [4].

Так само, Закон про захист конфіденційності споживачів Каліфорнії (CCPA) встановлює суворі правила щодо збору персональних даних. Згідно з CCPA,

споживачі можуть видаляти свою особисту інформацію, відмовитися від продажу даних та мати право на захист від дискримінації за здійснення цих прав [4].

Обидва закони наголошують на необхідності згоди та прозорості обробки персональних даних, впливаючи на те, як компанії підходять до веб-скрапінгу та використання даних. Без дотримання цих правил підприємства можуть зіткнутися з юридичними наслідками [4].

Щоб зрозуміти, чи є веб-скрапінг законним, давайте розглянемо реальні приклади. Ці справи можуть показати нам поточний стан галузі та куди вона може рухатися. Ось деякі з найвідоміших справ. Пам’ятайте, що це лише приклади. Завжди звертайтеся за професійною консультацією у вашій ситуації.

Ryanair проти PR Aviation (2018). Ryanair подала до суду на PR Aviation за скрапінг цін на авіаквитки, стверджуючи про порушення їхніх Умов надання послуг (ToS). Суд розглянув, чи становили Умови надання послуг Ryanair, які були представлені як угода browsewrap (посилання на умови внизу сторінки), обов’язковий договір [4].

Нідерландський суд постановив, що оскільки PR Aviation прямо не погодилася з цими умовами, дійсний договір не був укладений. Ця справа підкреслює правову складність угод browsewrap та підкреслює важливість чітких, виконавчих Умов надання послуг для діяльності з веб-скрапінгу. Ryanair виграла цю справу [4].

HiQ Labs збирала публічні дані з профілів LinkedIn для аналітики робочої сили, що призвело до того, що LinkedIn видала лист про припинення та утримання. HiQ звернулася до суду з проханням про вирішення цієї проблеми, стверджуючи, що вилучення публічних даних є законним. Суд став на бік HiQ, заявивши, що доступ до публічних профілів не порушує Закон про комп'ютерне шахрайство та зловживання (CFAA) [4].

У цій справі наголошується на відмінності між публічними та приватними даними та підкреслюється, що вилучення публічно доступної інформації, якщо воно здійснюється прозоро, може не порушувати федеральні закони. Однак це

рішення також наголосило на необхідності чітких рекомендацій та етичних практик у зборі даних [4].

1.2.3 Етичні принципи та рекомендації щодо використання веб-скрапінгу

Перш ніж розпочинати будь-який веб-скрапінг, варто звернутися за юридичною консультацією. Ось кілька ключових порад щодо етичного та відповідного веб-скрапінгу [4].

Використовуйте API, якщо вони доступні: Багато веб-сайтів пропонують API для збору даних. Вони є кращим вибором, ніж скрапінг [4].

Дотримуйтесь Умов обслуговування (ToS) сайту: Завжди читайте та поважайте Умови обслуговування веб-сайту, який ви хочете скрапувати [4].

Перевірте robots.txt: Цей файл показує, які частини сайту ви можете скрапувати. Якщо скрапінг заборонено, подумайте про те, щоб запитати дозволу у власника сайту [4].

Поважайте закони про авторське право: Переконайтеся, що дані, які ви скрапуєте, не захищені авторським правом. Якщо вам потрібно використовувати дані, захищені авторським правом, отримайте письмовий дозвіл від власника [4].

Веб-скрапінг займає складне правове поле, на яке впливають умови надання послуг, закони про авторське право та правила конфіденційності. Він не є незаконним за своєю суттю, але його законність залежить від дотримання певних правил та рекомендацій. Ключові справи, такі як Meta проти Bright Data та X проти Bright Data, ілюструють зміну законів про веб-скрапінг та підкреслюють важливість розуміння та дотримання цих правил [4].

Дотримуючись передового досвіду, залишаючись в курсі правових оновлень та звертаючись за юридичною консультацією, коли це необхідно, компанії можуть етично та ефективно використовувати веб-скрапінг, щоб отримати цінну інформацію, мінімізуючи юридичні ризики [4].

1.3 Аналіз методів визначення популярних товарних підкатегорій та прогнозування їх популярності

Щоб зрозуміти, як дані змінюються з часом, використовується аналіз та прогнозування часових рядів, які допомагають відстежувати минулі закономірності та прогнозувати майбутні значення. Цей аналіз:

– широко використовується у фінансах, погоді, продажах та даних датчиків;

– зосереджується на даних, зібраних через регулярні проміжки часу;

– допомагає визначити тенденції, сезонність та раптові зміни;

– корисний для планування, прогнозування та прийняття рішень;

– поширені методи включають ARIMA, експоненціальне згладжування та моделі машинного навчання [5].

Часові ряди (time series) – це послідовність спостережень певного показника, які впорядковані в часі. Найчастіше їх візуалізують за допомогою лінійного графіка, де по осі X відкладається час (дні, місяці, роки тощо), а по осі Y – значення досліджуваного показника. Така візуалізація дозволяє легко побачити тренди, коливання та приховані закономірності у даних. Завдяки цьому дослідники та аналітики можуть зрозуміти, як змінюється показник з часом і які фактори можуть впливати на ці зміни [5].

Аналіз часових рядів дозволяє будувати моделі, які допомагають прогнозувати майбутні значення показників на основі історичних даних [5]. Це особливо важливо для бізнесу та економіки, де прогнозування використовується для оцінки майбутнього попиту на товари, прогнозування доходів компанії, змін цін на ринку або динаміки фінансових активів. Завдяки таким прогнозам організації можуть заздалегідь планувати свої ресурси та приймати більш обґрунтовані рішення.

Часові ряди допомагають при знаходженні циклічних та сезонних закономірностей [5]. Наприклад, попит на певні товари може зростати у

святкові періоди або змінюватися залежно від пори року. Крім того, аналіз дозволяє виявляти аномалії – незвичайні або неочікувані зміни у даних, які можуть свідчити про помилки вимірювання, технічні збої або важливі події на ринку.

Завдяки аналізу часових рядів можна виявляти ранні сигнали потенційних проблем або небезпечних тенденцій [5]. Наприклад, різке падіння продажів або незвичайні зміни в поведінці користувачів можуть бути сигналом необхідності швидкого реагування. Раннє виявлення таких змін дозволяє організаціям запобігати фінансовим втратам або іншим негативним наслідкам.

Результати аналізу часових рядів активно використовуються у стратегічному плануванні [5]. На їх основі компанії можуть планувати бюджети, оптимізувати кількість персоналу, керувати запасами товарів та визначати довгострокову політику розвитку. Наприклад, прогноз попиту допомагає уникнути як дефіциту товарів, так і надлишкових запасів.

Організації, які ефективно використовують аналіз часових рядів, можуть швидше реагувати на зміни ринку та адаптувати свої стратегії [5]. Це дає змогу оптимізувати операційні процеси, краще задовольняти потреби клієнтів і випереджати конкурентів у прийнятті рішень.

Часові ряди можна групувати на основі структури, часового інтервалу та статистичної поведінки. Ці категорії допомагають у виборі правильного методу прогнозування [5].

1. Одновимірний проти багатовимірного. Одновимірний часовий ряд реєструє лише одну змінну з плином часу, що спрощує його аналіз та моделювання. Багатовимірний часовий ряд відстежує кілька пов'язаних змінних разом, щоб показати, як вони впливають одна на одну з плином часу [5].

2. Безперервний проти дискретного часового ряду. Безперервний часовий ряд спостерігається в кожен момент або з дуже високою частотою, як-от сигнали ЕКГ або датчиків. Дискретний часовий ряд реєструється з

фіксованими інтервалами, такими як щогодини, щодня або щомісяця, і є найпоширенішим форматом [5].

3. Стаціонарний проти нестаціонарного. Стаціонарний ряд має постійне середнє значення, дисперсію та закономірність з плином часу без тренду чи сезонності. Нестаціонарний ряд показує змінні закономірності, такі як тренди чи сезонність, і часто потребує трансформації перед моделюванням [5].

У контексті аналізу популярності товарних підкатегорій, дані про популярність товарів збираються через певні регулярні проміжки часу (наприклад, щодня або щомісяця), тому такі дані належать до дискретних часових рядів.

Крім того, для оцінювання популярності окремої товарної підкатегорії у часі зазвичай аналізується один основний показник (наприклад, кількість переглядів, продажів або сформований індекс популярності). У такому випадку часовий ряд можна розглядати як одновимірний, оскільки він відображає зміну одного показника з плином часу.

Водночас популярність товарів у сфері електронної комерції часто змінюється під впливом сезонних факторів, модних тенденцій та поведінки споживачів, що призводить до появи трендів та сезонних коливань у даних. Тому такі часові ряди зазвичай є нестаціонарними, оскільки їх статистичні характеристики можуть змінюватися з часом.

Таким чином, для дослідження, присвяченого аналізу методів визначення популярних товарних підкатегорій та прогнозування їх популярності, доцільно використовувати одновимірні дискретні нестаціонарні часові ряди, які найкраще відображають динаміку зміни популярності товарів у часі та дозволяють застосовувати відповідні методи прогнозування.

Попередня обробка даних часових рядів є одним з важливих кроків перед тим, як приступити до аналізу.

Попередня обробка часових рядів включає очищення, перетворення та підготовку даних для аналізу або прогнозування. Головна мета – покращити якість даних, видалити шум та зробити ряд придатним для моделювання [5]. Звичайні завдання попереднбої обробки включають:

– обробка відсутніх значень: заповнення або інтерполяція відсутніх спостережень для підтримки безперервності;

– обробка викидів: визначення та виправлення екстремальних значень, які можуть спотворювати аналіз;

– стаціонарність та перетворення: застосування таких методів, як диференціювання, детрендування або десезоналізація, для стабілізації середнього значення та дисперсії з часом;

– масштабування та нормалізація: стандартизація даних для покращення продуктивності моделі;

– стабілізація дисперсії: застосування перетворень для зменшення мінливості та покращення передбачуваності [5].

Методи попередньої обробки часових рядів:

– стаціонарність: зберігає середнє значення та дисперсію постійними з часом;

– диференціювання: видаляє тренди шляхом віднімання попередніх значень;

– ковзна середня (MA): згладжує дані за допомогою середнього з фіксованим вікном;

– експоненціальна ковзна середня (EMA): зважена середня, що надає більшої важливості останнім даним;

– імпутація відсутніх значень: заповнює прогалини за допомогою інтерполяції або статистичних методів;

– виявлення та видалення викидів: визначає та коригує екстремальні значення;

– масштабування: коригує діапазон значень для порівнянності;

– нормалізація: стандартизує дані до загальної шкали [5].

Ще одним з кроків є візуалізація. Вона допомагає досліджувати, інтерпретувати та передавати інформацію з даних, що залежать від часу. Поширені методи візуалізації включають:

– лінійні діаграми;

– сезонні діаграми;

– гістограми та діаграми щільності;

– спектральний аналіз;

– діаграми декомпозиції [5].

Лінійні діаграми показують, як змінна змінюється з часом, допомагаючи легко виявляти тенденції, стрибки та коливання [5]. Приклад лінійної діаграми зображений на рисунку 1.1.

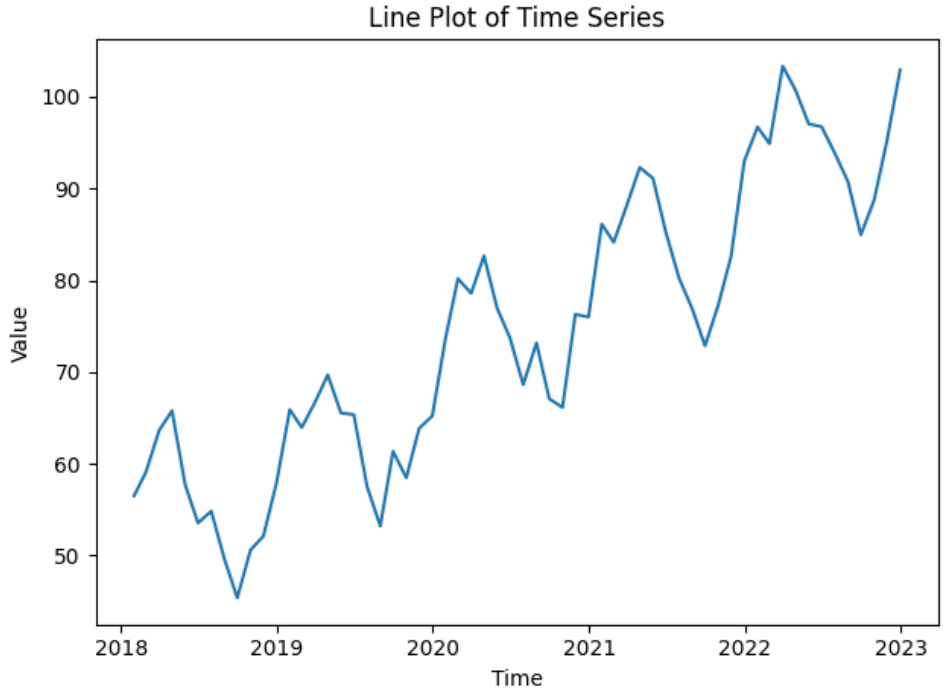


Рисунок 1.1 – Приклад лінійної діаграми [5]

Сезонні діаграми відображають повторювані закономірності протягом місяців, тижнів або сезонів, щоб виділити періодичну поведінку [5]. Приклад сезонної діаграми зображений на рисунку 1.2.

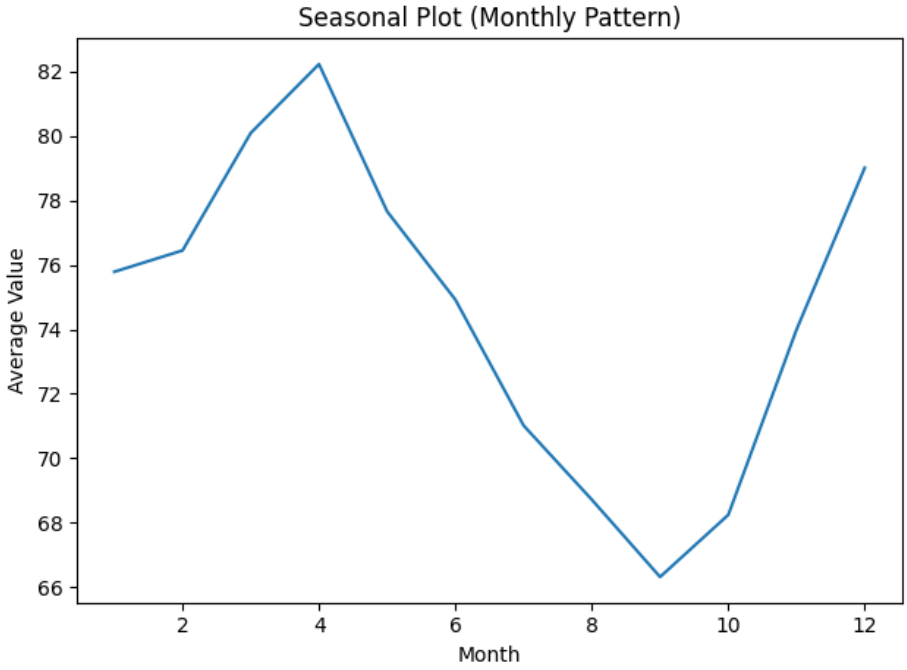


Рисунок 1.2 – Приклад сезонної діаграми [5]

Гістограми та діаграми щільності показують розподіл значень у часовому ряді, що полегшує розуміння загальних діапазонів та розкиду [5]. Приклад гістограми та діаграми щільності зображений на рисунку 1.3.

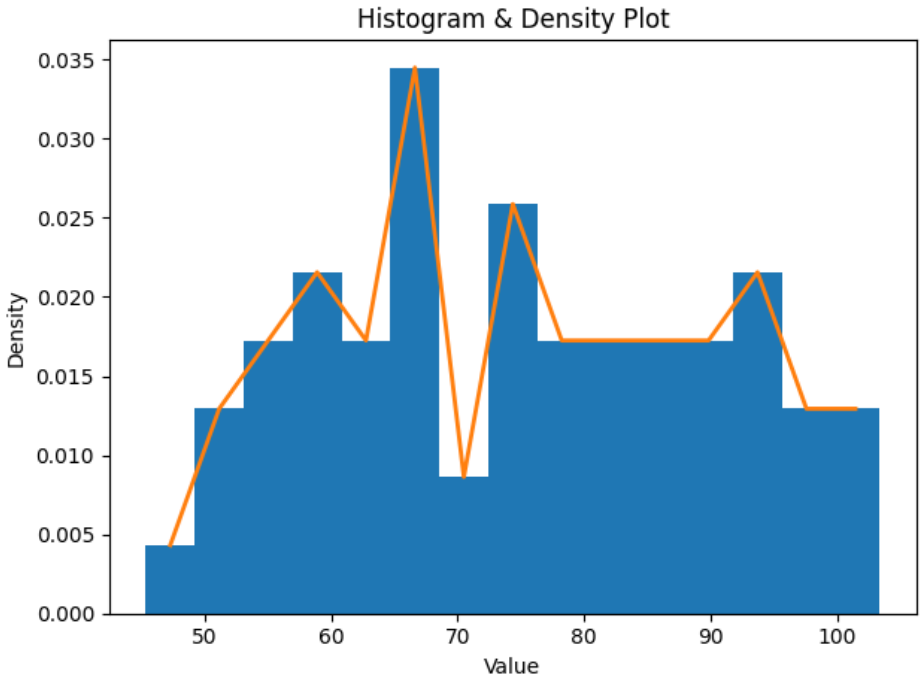


Рисунок 1.3 – Приклад гістограми та діаграми щільності [5]

Діаграми автокореляції та часткової автокореляції: ACF та PACF показують, як минулі значення впливають на поточні значення, допомагаючи вибрати відповідні моделі прогнозування [5]. Приклад діаграм автокореляції та часткової автокореляції зображений на рисунку 1.4.

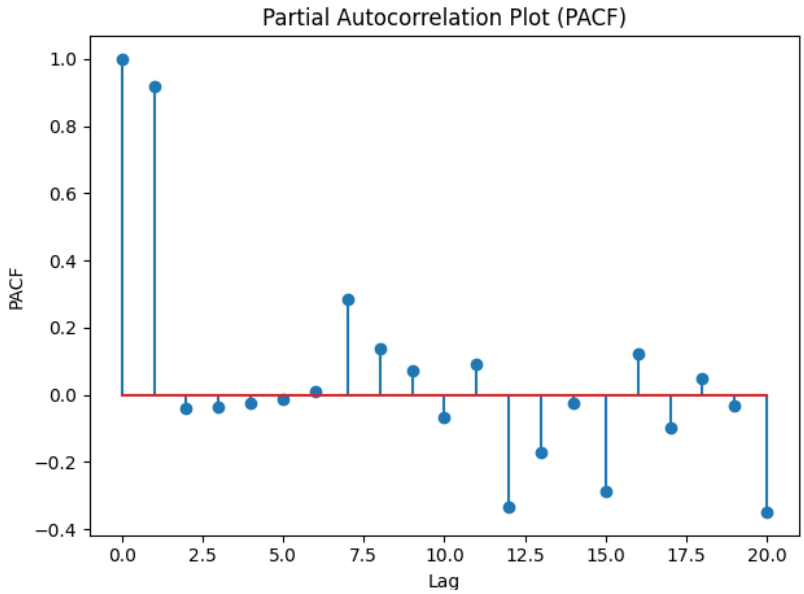


Рисунок 1.4 – Приклад діаграм автокореляції та часткової автокореляції [5]

Спектральний аналіз визначає домінуючі частотні цикли в даних, корисний для виявлення повторюваних довгострокових закономірностей [5]. Приклад спектрального аналізу зображений на рисунку 1.5.

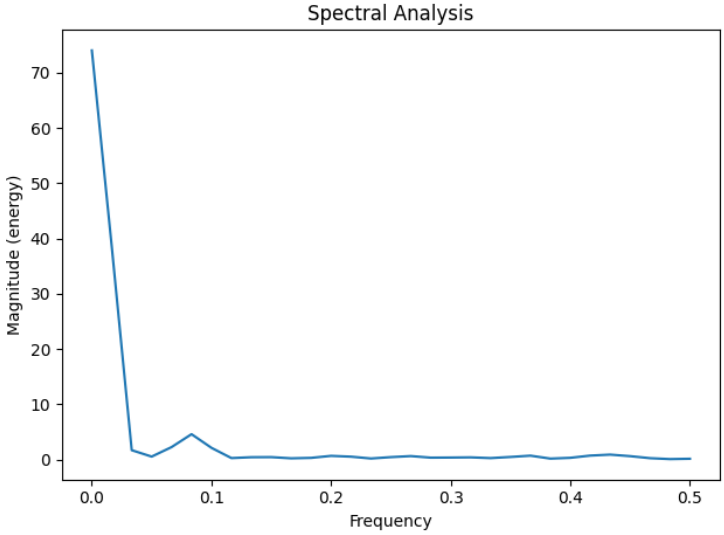


Рисунок 1.5 – Приклад спектрального аналізу [5]

Діаграми декомпозиції розділяють ряд на трендову, сезонну та залишкову частини, щоб чітко показати основну структуру [5]. Приклад діаграми декомпозиції зображений на рисунку 1.6.

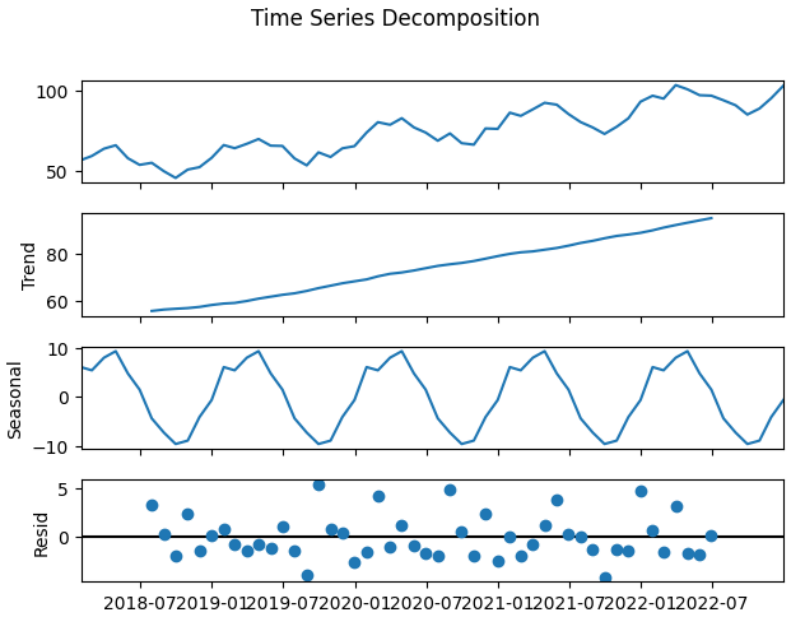


Рисунок 1.6 – Приклад діаграми декомпозиції [5]

Ці діаграми можна використати для аналізу отриманих результатів.

1.4 Актуальність обраного дослідження

Цифри описують актуальність будь якого дослідження набагато краще, ніж слова. Щоб отримати цифри, було прийняти рішення звернутися до популярного аналітичного сервісу Google Trends, який дозволяє оцінити актуальність теми, використовуючи пошукові слова.

Було обрано наступні пошукові фрази англійською мовою, щоб отримати якнайбільшу вибірку: web scraping та scraping, і також обрано в фільтрах розташуваня "Увесь світ". Це продемонстровано на рисунку 1.7.

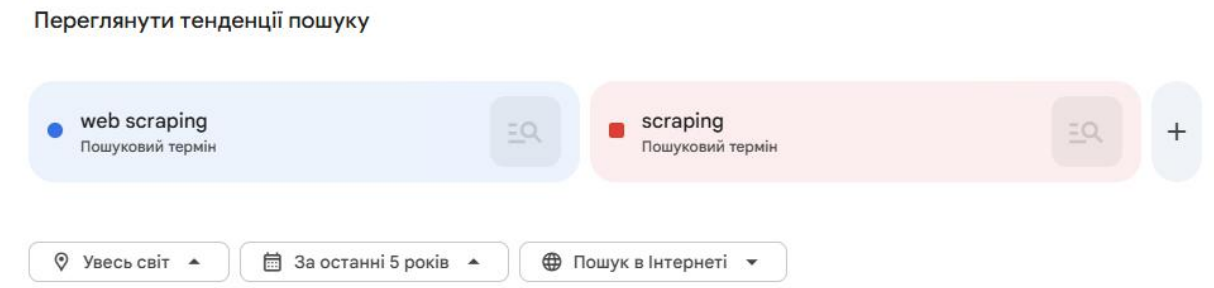


Рисунок 1.7 – Вибір ключових фраз для пошуку в Google Trends

Результат продемонстрований на рисунку 1.8.

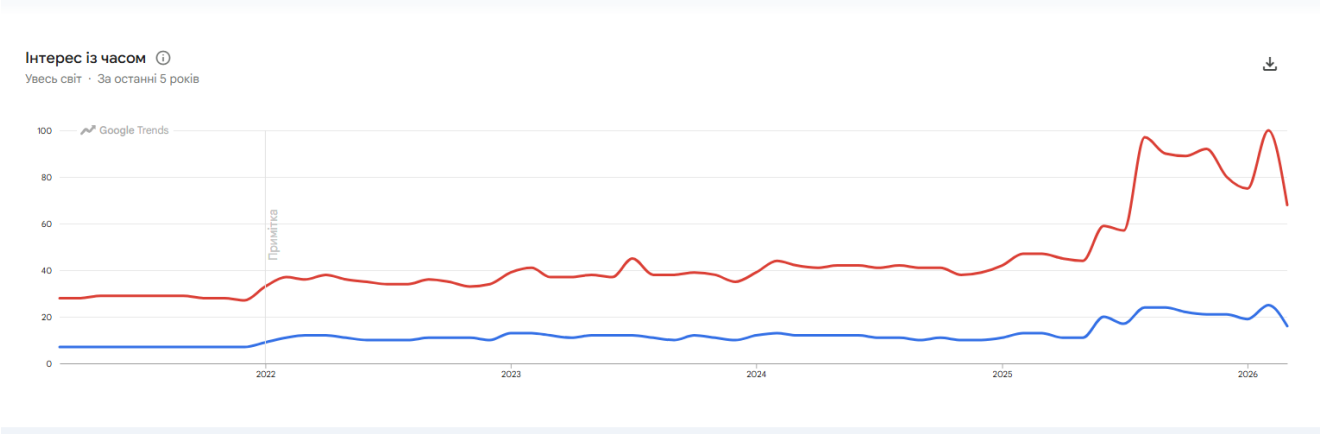


Рисунок 1.8 – Результат пошуку в Google Trends по запитах "web scraping" та "scraping"

Рівень пошукової зацікавленості за певний період часу, відображений на відносній шкалі від 0 до 100, де 100 означає найвищий рівень зацікавленості за вибраний період. Значення 50 указує на половину пікової популярності, а 0 – на недостатність даних [6].

Ми можемо спостерігати, що пошук зі словом "scraping" (скрапінг) досягнув піку в лютому 2026, отримавши максимальний показник 100. Навіть незважаючи на те, що це відносні дані, вони вже відображають різке збільшення зацікавленості людей в скейпінгу, порівняно з тою, яка була протягом останніх п'яти років. В середині 2025-го року зацікавленість зі значення 35 почала рости, і

досягла максимально можливого піку в лютому 2026. Це приріст на +65%, що дуже показово.

Якщо зробити пошук по фразі "data analysis" (аналіз даних), ми побачимо майже ідентичну динаміку приросту зацікавленості до обраної теми дослідження, з тим самим піком в 100 балів у лютому 2026-го. Це продемонстровано на рисунку 1.9.

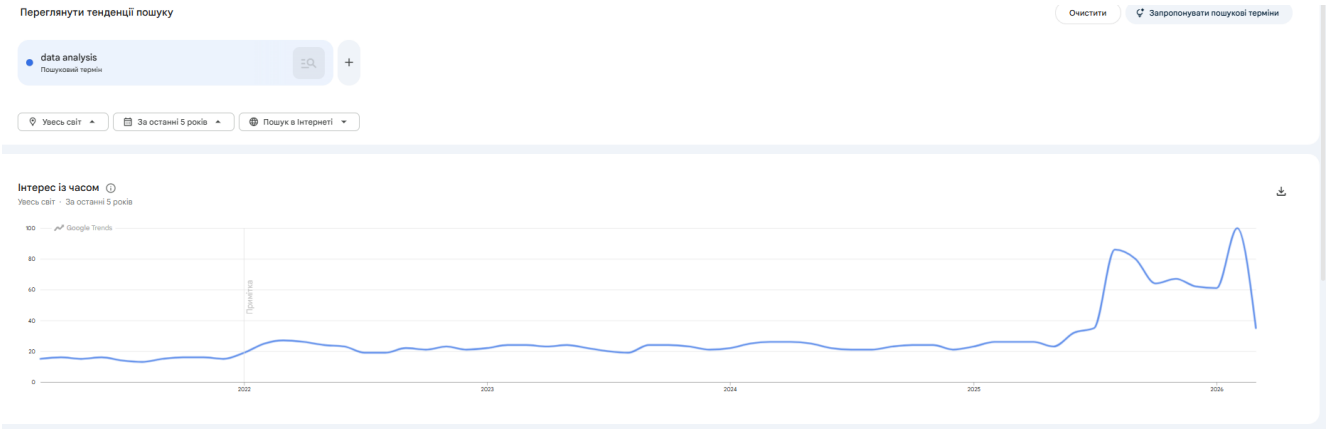


Рисунок 1.9 – Результат пошуку в Google Trends по запиту " data analysis"

Це доводить, що обрана тема дослідження є максимально актуальною у сьогоднішньому контексі області інформаційних технологій.

2 АНАЛІЗ МЕТОДІВ, СИСТЕМ ТА ПРОЄКТУВАННЯ

2.1 Обґрунтування вибору мови програмування Python

Вибір мови програмування є одним із ключових архітектурних рішень при розробці систем аналізу даних. У контексті дослідження, присвяченого аналізу методів визначення популярних товарних підкатегорій та прогнозуванню їх популярності на основі часових рядів, обґрунтованим є застосування мови Python версії 3.10+. Нижче наведено системне обґрунтування цього вибору за ключовими критеріями.

Мова Python є фактичним стандартом у галузі наукових обчислень та аналізу даних, що підтверджується щорічними індексом популярності TIOBE [7].

Python розроблений з надзвичайно простим синтаксисом разом із потужним вбудованим ядром. Це поєднання сприяє швидкій розробці програмного забезпечення, що робить Python привабливим вибором для досвідчених розробників програмного забезпечення. Зовсім недавно Python став мовою для викладання інформатики. Ті самі властивості, які роблять Python популярним серед досвідчених програмістів, такі як простота та надійність, роблять його привабливим як мову навчання для початківців. Наш власний перехід на Python був заснований на бажанні витрачати набагато більше часу на вивчення основних принципів і набагато менше зусиль, відволікаючихся на штучні перешкоди мови програмування. Як об'єктно-орієнтована мова, Python пропонує простішу та послідовнішу модель, ніж інші традиційні об'єктно-орієнтовані мови, такі як Java та C++ [8].

Ключовою перевагою є наявність зрілих спеціалізованих бібліотек, які безпосередньо застосовані в розробленій системі і продемонстровані в таблиці 2.1:

Таблиця 2.1 – Використані бібліотеки

Бібліотека	Призначення в системі
pandas	Зберігання та маніпуляція часовими рядами
statsmodels	Статистичний аналіз, декомпозиція, прогнозування
numpy	Чисельні обчислення та трансформації
matplotlib	Візуалізація результатів аналізу
aiohttp	Асинхронні HTTP-запити при зборі даних
asyncpg	Асинхронна взаємодія з PostgreSQL
beautifulsoup4	Парсинг HTML-сторінок інтернет-магазинів

Використання вказаних бібліотек дозволяє реалізувати повний аналітичний конвеєр – від збору сирих даних до побудови прогнозів – без необхідності розробки низькорівневих алгоритмів з нуля. Це суттєво скорочує час розробки та підвищує відтворюваність результатів дослідження.

Python містить вбудований модуль asyncio, що реалізує модель конкурентного виконання на основі циклу подій (event loop). У розробленій системі асинхронність є критично важливою з двох причин. По-перше, під час збору даних необхідно одночасно обробляти до декількох десятків HTTP-з'єднань до сайтів інтернет-магазинів. По-друге, запити до бази даних PostgreSQL виконуються асинхронно, що виключає блокування основного потоку виконання.

Реалізація в коді: клас Scraper використовує asyncio.Semaphore для обмеження кількості одночасних запитів та asyncio.gather для паралельного

виконання задач збору даних, що реалізує класичний патерн producer-consumer у межах одного процесу Python.

Мова Python поєднує гнучкість динамічної типізації з можливістю явного оголошення типів через модуль typing та вбудовані анотації (PEP 484). У розробленій системі всі публічні методи та параметри функцій містять повні анотації типів (наприклад, pd.Series, list[dict[str, Any]], tuple[str | None, int | None]), що дозволяє використовувати статичні аналізатори типів (mypy, pyright) та покращує читабельність коду. Директива from __future__ import annotations присутня у всіх модулях системи та забезпечує відкладене обчислення анотацій.

Розроблена система взаємодіє з реляційною СУБД PostgreSQL через бібліотеку asyncpg, яка надає нативний асинхронний драйвер без залежності від синхронного psycopg2. Це забезпечує максимальну продуктивність при записі результатів аналізу в таблиці бази даних. Одночасно бібліотека aiohttp забезпечує HTTP-клієнт промислового рівня з підтримкою timeouts, автоматичного повторного надсилання запитів та сесійного управління заголовками. Така комбінація інструментів нативно підтримується Python і потребує мінімального коду налаштування.

2.2 Застосовані об'єктно-орієнтовані технології та принципи

Архітектура розробленої системи базується на принципах об'єктно-орієнтованого програмування (ООП).

Основною перевагою об'єктно-орієнтованого програмування є те, що воно забезпечує основу для моделювання великої складної програмної системи як сукупності окремих компонентів. Це може допомогти в початковому проектуванні та впровадженні програмного забезпечення, в подальшому обслуговуванні цього програмного забезпечення та в повторному використанні окремих компонентів в інших контекстах. З цих причин об'єктно-орієнтоване програмування стало домінуючим стилем у галузі для масштабної розробки програмного забезпечення [8].

Ключем до гарного об'єктно-орієнтованого проектування є виявлення та моделювання спільних рис та відмінностей між сутностями. Коли ми починаємо розглядати моделювання різних класів, нам слід враховувати, як ці класи порівнюються один з одним. Визначивши спільні риси, ми можемо організувати ієрархію класів [8].

Ознайомимося з основними принципами об'єктно-орієнтованого програмування – однієї з основних методологій програмування, яка базується на ідеї, що програма – це кластер об'єктів, кожен з яких належить до певного класу, а класи утворюють ієрархію успадкування [9].

Інкапсуляція. Під час розробки компонента програми один з основних принципів полягає в:

1. Приховуванні внутрішніх даних та деталей реалізації компонента від інших компонентів програми.

2. Наданні набору методів для взаємодії з компонентом (API).

3. Цей принцип є одним із чотирьох фундаментальних принципів ООП і називається інкапсуляцією [9].

Правильна інкапсуляція важлива з багатьох причин:

1. Вона дозволяє повторно використовувати компоненти. Компоненти можна використовувати в ширшому контексті, оскільки вони взаємодіють один з одним лише через API та не чутливі до змін внутрішньої структури.

2. Інкапсуляція прискорює процес розробки. Компоненти, слабо пов'язані один з одним (це означає компоненти, код яких викликає інші компоненти або використовує їхній код якомога рідше), можна розробляти, тестувати та доповнювати незалежно.

3. Правильно інкапсульовані компоненти легше зрозуміти та налагодити, що полегшує підтримку програми [9].

Існує чотири модифікатори доступу:

– public – повний доступ до сутності (поля/методу класу) з будь-якого пакета;

– protected – доступ до сутності мають лише класи в пакеті та дочірні класи;

– implicit – доступ до сутності мають лише класи в пакеті;

– private – доступ лише в класі, в якому оголошено сутність [9].

Для коректної інкапсуляції нам також потрібен коректний API для роботи з компонентом. Наприклад, ми можемо включити логіку перевірки значень, що передаються змінній, або навіть не надавати сетери в класі, якщо клас має бути доступний лише для читання [9].

Успадкування. Успадкування є одним з основних принципів об'єктно-орієнтованого програмування, оскільки воно дозволяє побудувати ієрархічну структуру об'єкта. За допомогою успадкування можна створити загальний клас, який визначатиме характеристики та поведінку для деякого кластера пов'язаних об'єктів. Пізніше цей клас може успадковуватися іншими, більш приватними класами, кожен з яких додає свої унікальні характеристики та таким чином доповнює та змінює поведінку базового класу. У термінології Java такий загальний клас називається суперкласом, або базовим класом, або батьківським класом, а клас, який його успадковує, називається підкласом, або дочірнім класом, або похідним класом [9].

Успадкування забезпечує зв'язок "є-а" між суперкласом та підкласом. Наприклад, є клас "Співробітник" та клас "Менеджер". Кожен менеджер також є співробітником компанії, тому існує зв'язок "є-а" між класом "Менеджер" та класом "Співробітник". Таким чином, з точки зору успадкування, "Співробітник" є суперкласом, а "Менеджер" – підкласом. Більше того, клас, який успадковує якийсь інший клас, може сам бути суперкласом для одного або кількох інших класів. Також, порівняно, наприклад, з C++, в Java, строго кажучи, немає множинного успадкування, що означає, що будь-який клас може мати не більше

одного суперкласу. І всі класи, для яких суперклас чітко не вказано, успадковують клас "Об'єкт" за замовчуванням [9].

Клас "Співробітник" з наведеного вище прикладу є суперкласом не тому, що він вищий за клас "Менеджер" або має більше функціональності. Навпаки: підкласи мають не менше, а часто набагато більше функціональності, ніж їхні суперкласи. Префікси "супер-" та "під-" запозичені з математики: масив усіх співробітників містить масив усіх менеджерів. Таким чином, масив усіх менеджерів є підмасивом масиву співробітників [9].

Поліморфізм. Говорячи про поліморфізм, слід пам'ятати, що цей принцип нерозривно пов'язаний з іншим принципом ООП – успадкуванням, яке допомагає реалізувати поліморфізм [9].

Візьмемо як приклад абстрактний клас "Автомобіль", який успадковує два специфічні класи: "Спортивний автомобіль" та "Вантажівка" [9].

Як спортивні автомобілі, так і вантажівки мають спільні характеристики та можуть виконувати ті ж дії, що й інші автомобілі. Ці дії вказані в абстрактному суперкласі, але їх можна виконувати по-різному [9].

Наприклад, дія "запустити двигун" є спільною для всіх автомобілів, але ви можете виконати її, натиснувши кнопку у випадку зі спортивним автомобілем та повернувши ключ запалювання у випадку з вантажівкою. Різні рішення – один результат. Ось що таке поліморфізм [9].

Якщо бути більш конкретним, поліморфізм – це один з принципів ООП, який дозволяє викликати перевизначений метод через змінну батьківського класу та таким чином провокувати поведінку, яка відповідала б реальному підкласу, до якого посилається ця змінна [9].

Абстракція. З відносно недавніх часів абстракцію виділяють як самостійний четвертий принцип [9].

Одне з основних визначень слова "абстракція", яке можна зустріти в сучасних словниках. Абстракція – це процес видалення фізичних, просторових або часових деталей чи атрибутів при вивченні об'єктів чи систем, щоб зосередити увагу на деталях більшої важливості [9].

Усі мови програмування надають певні абстракції. Наприклад, мови сімейства асемблерів є певним видом абстракції відповідних мікропроцесорів, оскільки вони дозволяють нам відвернути нашу увагу від деталей їхньої реалізації та взаємодіяти з ними через певний набір інструкцій вищого рівня. Імперативні мови програмування, що з'явилися після асемблера, наприклад Basic, Fortan, C, були абстракцією вищого рівня порівняно з мовами асемблера – вони дозволяли використовувати деякий синтаксис, більш звичний людині, роблячи синтаксис ближчим до природних мов [9].

Об'єктно-орієнтовані мови, такі як Java, надають ще вищий рівень абстракції: в ООП об'єкти є моделями реальних понять, таких як "співробітник", "сервер", "запис у щоденнику". Акцент робиться лише на тих властивостях цих понять, які необхідні в певному випадку для вирішення певної задачі [9].

Наприклад, у застосунку для відстеження студентів університету клас "Студент", окрім загальних полів, таких як ім'я, прізвище, дата народження тощо, містить поля з інформацією про номер залікової книжки, статус студента (активний, у академічній відпустці, відрахований), викладацький склад, оцінки за семестри тощо. Але така інформація не має значення для того ж класу "Студент" у застосунку для відстеження студентів. У цьому випадку клас містить поля, що відповідають навчальним проектам, до яких призначені студенти, їхньому рівню англійської мови за результатами останніх тестів, кількості відвіданих заходів тощо [9].

Ось що таке абстракція: певний об'єкт повинен виконувати певні завдання, а розробник зосереджений на властивостях об'єкта, що стосуються цих завдань. Як наслідок, в об'єктно-орієнтованих мовах розробники зосереджені на термінах предметної області, для якої вони розробляють застосунок, на відміну від імперативних мов, де розробники повинні зосередитися на термінах комп'ютерної логіки [9].

SOLID – це п'ять основних принципів, які допомагають у створенні гарної архітектури програмного забезпечення. SOLID – це абревіатура, де S означає SRP (принцип єдиної відповідальності), O – OCP (принцип відкрито-закритого

принципу), L – LSP (принцип заміщення Ліскова), I – ISP (принцип розділення інтерфейсів), D – DIP (принцип інверсії залежностей) [10].

Використовуючи ці принципи SOLID, ми можемо створювати ефективне, багаторазове та некрихке програмне забезпечення, яке є сталим та придатним для підтримки в довгостроковій перспективі. Можливість повторного використання, розширюваність, придатність для підтримки та стійкість (RESM) – це основні питання, пов'язані з функціональним програмуванням. Щоб подолати ці проблеми та створити динамічне програмне забезпечення, нам потрібні принципи SOLID. Принципи SOLID дали відповідну відповідь для розробки ефективної архітектури програмного забезпечення, яка може подолати проблеми RESM [10].

Принцип єдиної відповідальності (SRP) Принцип єдиної відповідальності стверджує, що клас має бути розроблений для виконання однієї функції або обов'язку. Цей принцип є життєво важливим, оскільки він гарантує, що кожен клас залишається присвяченим певному завданню, що, у свою чергу, спрощує його розуміння, підтримку та потенціал для розширення [11].

SRP Принцип єдиної відповідальності (SRP) спрощує код, гарантуючи, що кожен клас розроблений з окремою та специфічною функцією. Такий цілеспрямований підхід підвищує читабельність та зручність підтримки коду. Коли класи відповідають SRP, їх стає простіше рефакторувати, тестувати та розширювати. Обмежуючи кожен клас однією відповідальністю, модифікації з меншою ймовірністю вплинуть на інші частини системи, тим самим мінімізуючи ризик виникнення помилок [11].

Принцип відкритості/закритості стверджує, що програмні компоненти, такі як класи, модулі або функції, повинні бути розроблені таким чином, щоб дозволяти розширення без необхідності змін до їхнього існуючого коду. Цей принцип виступає за покращення функціональності модуля за рахунок нового коду, а не модифікацій існуючої кодової бази [11].

OCP має вирішальне значення для підтримки стабільності програмного забезпечення в міру його розвитку. Дотримуючись OCP, розробники можуть

додавати нові функції до існуючого коду, не змінюючи існуючу кодову базу, мінімізуючи ризик появи нових помилок [11].

Принцип заміщення Ліскова стверджує, що екземпляри батьківського класу повинні бути замінюваними екземплярами похідного класу без зміни коректності програми. Цей принцип є вирішальним для підтримки добре структурованої ієрархії класів, гарантуючи, що підкласи можуть безперешкодно замінювати свої батьківські класи без внесення помилок або невідповідностей [11].

LSP гарантує, що підклас може замінити свій суперклас, роблячи код більш передбачуваним та надійним. Порушення LSP може призвести до неочікуваної поведінки в програмному забезпеченні, особливо коли підкласи перевизначають методи способами, які не узгоджуються з передбачуваною поведінкою суперкласу. LSP має вирішальне значення для правильного використання поліморфізму та успадкування. Він гарантує, що похідний клас може бути замінений на його базовий клас без зміни поведінки програми. Дотримання LSP призводить до більш надійного та зручного в обслуговуванні коду, особливо у великих системах, де поліморфізм активно використовується [11].

Принцип сегрегації інтерфейсів радить, що жоден клієнт не повинен бути змушений залежати від методів, які він не використовує. Цей принцип сприяє створенню менших, більш специфічних інтерфейсів, а не великих, універсального призначення.

ISP сприяє роз'єднанню, гарантуючи, що класи залежать лише від інтерфейсів, які їм релевантні. Це зменшує вплив змін і робить кодову базу більш гнучкою. Дотримання ISP може призвести до більш модульного та тестованого коду. Дотримуючись ISP, ми створюємо більш сфокусовані та простіші в обслуговуванні інтерфейси. Це зменшує ризик виникнення змін, що призводять до їхньої еволюції, та підвищує модульність кодової бази. ISP також спрощує тестування, оскільки кожен клас можна тестувати окремо від методів, які він не використовує.

Принцип інверсії залежностей диктує, що високорівневі компоненти не повинні залежати від низькорівневих компонентів; натомість, обидва повинні

залежати від абстракцій. Крім того, він наголошує на тому, що абстракції не повинні спиратися на конкретні деталі, а ці деталі повинні залежати від абстракцій [11].

Принцип інверсії залежностей (DIP) виступає за розділення програмних компонентів, пропагуючи залежність від абстракцій, а не від конкретних реалізацій. Такий підхід значно підвищує модульність коду, роблячи розробку, тестування та обслуговування набагато легшими. Розділюючи високорівневі та низькорівневі модулі за допомогою абстракцій, системи стають більш гнучкими та стійкими до змін, що дозволяє легко оновлювати та замінювати з мінімальним порушенням загальної архітектури. Впровадження DIP допомагає у створенні надійної та адаптивної структури програмного забезпечення, яка може враховувати мінливі вимоги та технологічні досягнення, зрештою забезпечуючи кращу масштабованість та довговічність програми [11].

У системі визначено п'ять основних класів, кожен з яких відповідає за чітко визначений аспект функціональності:

Таблиця 2.2 – Класи системи

Клас	Файл	Відповідальність (принцип SRP)
TimeSeriesAnalyzer	analysis.py	Повний конвеєр статистичного аналізу часових рядів
TimeSeriesVisualizer	visualizer.py	Генерація matplotlib-фігур з результатів аналізу
Scraper	scraper.py	Асинхронний збір даних з веб-сторінок
ProductParser	parser.py	Парсинг HTML-сторінки одного товару
ListingParser	parser.py	Парсинг сторінки-каталогу та пагінації
Database	database.py	Асинхронна взаємодія з PostgreSQL

Приклад інкапсуляції: у класі TimeSeriesAnalyzer публічний інтерфейс зведено до двох методів (run та run_many), тоді як уся реалізація – тестування стаціонарності, декомпозиція, побудова кандидатів моделей SARIMA, збереження результатів – схована у приватних методах з префіксом підкреслення (_make_stationary, _decompose, _fit_and_forecast, _persist тощо).

Декоратор @dataclass автоматично генерує методи __init__, __repr__ та __eq__ на основі оголошених полів з анотаціями типів. У системі dataclass-и використовуються для двох категорій об'єктів:

1. Контейнери результатів аналізу (файл analysis.py):

– StationarityResult – зберігає результати тестів ADF та KPSS, перелік застосованих трансформацій та рекомендацію;

– DecompositionResult – зберігає компоненти STL-декомпозиції (trend, seasonal, residual, observed) та параметри методу;

– ForecastResult – зберігає прогнозні значення, межі довірчого інтервалу, ідентифікатор обраної моделі та значення AIC;

– AnalysisResult – агрегує всі три вищезазначені результати разом із вихідним та трансформованим рядом.

2. Конфігураційні об'єкти (файл scraper.py):

– ScraperConfig – конфігурація скрапера: URL, рівень конкурентності, таймаут, параметри повторних спроб, назва таблиці БД;

– ScrapeResult – результат обробки одного URL: статус-код, розібрані елементи, журнал подій, повідомлення про помилку;

– ListingPage – результат парсингу сторінки-каталогу: список URL товарів, посилання на наступну сторінку, загальна кількість товарів.

Використання dataclass замість звичайних словників (dict) забезпечує чіткий контракт між компонентами системи: тип кожного поля задокументований статично, а не виводиться неявно з коду, що спрощує підтримку та рефакторинг.

Композиція – механізм побудови складних об'єктів шляхом включення екземплярів інших класів як атрибутів – є пріоритетним підходом у розробленій

архітектурі. Клас Scraper не успадковує функціональність парсера чи бази даних, а приймає їх як залежності через конструктор.

Такий підхід реалізує патерн Dependency Injection (ін'єкція залежностей): класи отримують свої залежності ззовні, а не створюють їх самостійно. Це дозволяє замінювати реалізації (наприклад, підставити тестову базу даних або альтернативний парсер) без модифікації коду класу-споживача.

Протокол контекстного менеджера в Python (методи __enter__ та __exit__) гарантує коректне вивільнення ресурсів навіть у разі виникнення винятків. Для асинхронного коду передбачено аналогічний протокол: __aenter__ та __aexit__.

У розробленій системі цей механізм реалізовано у двох класах: Database та Scraper. Обидва класи підтримують синтаксис async with, що забезпечує автоматичне відкриття пулу з'єднань при вході у блок та їх коректне закриття при виході – незалежно від того, завершився блок успішно чи з помилкою.

Цей підхід унеможливлює витоки з'єднань (connection leaks) – типову проблему при роботі з пулами з'єднань до бази даних у довготривалих процесах.

Статичні методи (@staticmethod) використовуються для операцій, що логічно належать до класу, але не потребують доступу до стану екземпляра чи самого класу. У TimeSeriesAnalyzer статичними є обчислювально незалежні операції – _test_stationarity, _detect_period, _decompose. У TimeSeriesVisualizer статичними є всі методи малювання (_draw_series, _draw_rolling тощо), оскільки вони приймають осі matplotlib як аргументи і не звертаються до стану об'єкта-візуалізатора.

Метод класу (@classmethod) реалізує патерн фабричного методу (Factory Method): Scraper.from_env() є альтернативним конструктором, що зчитує параметри конфігурації зі змінних середовища та повертає повністю налаштований екземпляр класу. Це відокремлює логіку конфігурування від логіки роботи скрапера.

Кожен клас системи несе відповідальність лише за один аспект функціональності, що відповідає принципу SRP – першому з принципів SOLID.

Розподіл відповідальності простежується не лише між класами, але й між модулями:

– parser.py – виключно логіка розбору HTML, без мережевих запитів і без звернень до БД;

– scraper.py – виключно оркестрація мережевих запитів і координація парсера з базою даних;

– database.py – виключно абстракція над asyncpg: CRUD-операції та управління пулом;

– analysis.py – виключно статистичний аналіз: стаціонарність, декомпозиція, прогнозування;

– visualizer.py – виключно генерація графіків на основі готових результатів аналізу.

Такий розподіл забезпечує незалежну тестованість кожного компонента та спрощує супровід системи: зміна методу парсингу HTML не вимагає модифікацій у логіці аналізу часових рядів, і навпаки.

2.3 Логіка аналізу отриманих результатів

Ми визначили, що для дослідження доцільно використовувати одновимірні дискретні нестаціонарні часові ряди.

Тип даних – одновимірний дискретний часовий ряд. Наш результат скрапінгу це одновимірний дискретний часовий ряд. Метод _load_series завантажує з бази даних таблицю popularity_metrics, де кожен запис – це пара (period, value) для конкретної підкатегорії товарів та метрики (перегляди, продажі тощо). Результатом є об'єкт pd.Series з індексом типу DatetimeIndex – це і є одновимірний дискретний часовий ряд, де кожне спостереження прив'язане до конкретного моменту часу (доба, тиждень, місяць). Мінімальний розмір вибірки перевіряється явно: якщо точок менше 14 – аналіз не запускається.

Перевірка та досягнення стаціонарності (метод _make_stationary + _test_stationarity). Нестаціонарність є ключовою передумовою роботи з даними популярності товарів. Код застосовує двоетапний підхід.

Крок 1. Тестування стаціонарності двома незалежними тестами.

Тест ADF (Augmented Dickey-Fuller) – перевіряє нульову гіпотезу H_0 про наявність одиничного кореня (тобто ряд нестаціонарний). Якщо p-value < 0,05 – H_0 відхиляється, ряд вважається стаціонарним. Параметр autolag="AIC" означає, що кількість лагів підбирається автоматично за критерієм Акаіке.

Зазначений тест перевіряє нульову гіпотезу про наявність одиничного кореня в авторегресійній моделі, протиставляючи її альтернативній гіпотезі про стаціонарність ряду [12].

В основу тесту покладено авторегресійну модель першого порядку AR(1) такого вигляду [12]:

$$Y_t = \rho \cdot Y_{t-1} + \varepsilon_t \tag{2.1}$$

Дікі та Фуллер запропонували альтернативне формулювання, отримане відніманням Y_{t-1} від обох частин рівняння (2.1):

$$\begin{aligned} Y_t - Y_{t-1} &= \rho \cdot Y_{t-1} - Y_{t-1} + \varepsilon_t \\ \Delta Y_t &= (\rho - 1) \cdot Y_{t-1} + \varepsilon_t \\ \Delta Y_t &= \delta \cdot Y_{t-1} + \varepsilon_t \end{aligned} \tag{2.2}$$

де $\delta = \rho - 1$ [12].

Рівняння (2.2) є базовою формою тесту без константи. Окрім неї, Дікі та Фуллер запропонували дві розширені специфікації: модель з константою та модель з константою і детермінованим часовим трендом [12]:

$$\Delta Y_t = \alpha + \delta \cdot Y_{t-1} + \varepsilon_t \tag{2.3}$$

$$\Delta Y_t = \alpha + \beta t + \delta \cdot Y_{t-1} + \varepsilon_t \tag{2.4}$$

У всіх трьох формах перевіряються гіпотези: нульова H_0: $\delta = 0$ (ряд має одиничний корінь і є нестаціонарним) та альтернативна H_1: $\delta < 0$ (ряд не має одиничного кореня і є стаціонарним). Перевірка здійснюється за допомогою t-статистики [12]:

$$t = (\hat{y} - y_{h0}) / SE(\hat{y}) \quad (2.5)$$

Логіка прийняття рішення є такою: якщо розраховане значення t-статистики є більшим за відповідне критичне значення, нульова гіпотеза не відхиляється − ряд вважається нестаціонарним та містить одиничний корінь. Якщо ж розраховане значення t менше за критичне, нульова гіпотеза відхиляється і робиться висновок про стаціонарність ряду. В рамках графічної інтерпретації: якщо розраховане значення знаходиться праворуч від критичного на одностороньому хвості розподілу, нульова гіпотеза не відхиляється; якщо ліворуч − відхиляється. Альтернативно для прийняття рішення може використовуватися p-значення: при $p < 0,05$ нульова гіпотеза відхиляється.

Процедура тестування є ітеративною. Спочатку ряд перевіряється на рівні. У разі, якщо він не демонструє стаціонарності, перевірка послідовно виконується на першій різниці, а за необхідності − на другій різниці. Така покрокова процедура дозволяє визначити порядок інтегрованості ряду, що є необхідною передумовою для коректного вибору та специфікації моделі прогнозування.

Тест KPSS (Kwiatkowski–Phillips–Schmidt–Shin) – перевіряє протилежну гіпотезу: H_0 полягає в тому, що ряд є стаціонарним.

Нехай Y_i, $i = 1, \ldots, n$, буде спостережуваним часовим рядом, для якого необхідно перевірити стаціонарність, а $\bar{Y}_n$ − вибірковим середнім значенням Y_i. Розглянемо процес часткової суми відхилень від (оціненого) середнього значення процесу [13]:

$$S_n(u) = \Sigma_{i=1}^{u} (Y_i - \bar{Y}_n) \quad (2.6)$$

За нульової гіпотези стаціонарності виконується $E[S_n(u)] = 0$ для всіх $u = 1, \ldots, n$. Однак якщо процес є нестаціонарним, можна очікувати, що $E[S_n(u)]$ буде відмінним від нуля для деякого u. На підставі властивостей процесу часткової суми $S_n(u)$ Квятковський та ін. (1992) запропонували наступну статистику KPSS для перевірки стаціонарності [13]:

$$T_n = (1 / n^2 s_n^2) \cdot \Sigma_{u=1}^{n} [S_n(u)]^2 \tag{2.7}$$

де довгострокова дисперсія s_n^2 визначається як [13]:

$$s_n^2 = \Sigma_{i,j=1}^{n} w(|i-j|, m_n) \cdot \hat{\gamma}_{i-j},$$
$$\hat{\gamma}_j = (n-j)^{-1} \cdot \Sigma_{i=1}^{n-j} (Y_i - \bar{Y}_n)(Y_{i+j} - \bar{Y}_n) \text{ для } 0 \leq j < n \tag{2.8}$$

а m_n − пропускна здатність (bandwidth), що задовольняє умовам $m_n \to \infty$ та $m_n/n \to 0$. Величина s_n^2 є непараметричною оцінкою границі $nVar(\bar{Y}_n)$ за умови її існування [13].

За деяких м'яких регулярних умов Квятковський та ін. (1992) довели, що статистика T_n збігається за розподілом до інтегралу [13]:

$$\int_0^1 V(t)^2 \, dt \tag{2.9}$$

за умов стаціонарності, де V(t) − стандартний броунівський міст. Тест проводиться шляхом порівняння T_n з критичними значеннями верхнього хвоста розподілу $\int_0^1 V(t)^2 \, dt$, які апроксимуються чисельним моделюванням броунівського мосту V(t). Зокрема, якщо T_n перевищує бажане критичне значення верхнього хвоста, нульова гіпотеза стаціонарності відхиляється [13].

У розробленій системі тест KPSS реалізовано через функцію kpss бібліотеки statsmodels з параметром regression="c" (перевірка стаціонарності навколо константи) та автоматичним підбором кількості лагів (nlags="auto"). На відміну від тесту ADF, де нульова гіпотеза стверджує нестаціонарність, у тесті KPSS

нульова гіпотеза полягає у стаціонарності ряду − тому два тести є взаємодоповнювальними, і лише їхній узгоджений результат слугує підставою для остаточного висновку про характер досліджуваного часового ряду.

Якщо p-value > 0,05 – H_0 не відхиляється, тобто стаціонарність підтверджується. Регресія "c" означає перевірку стаціонарності навколо константи.

Ряд визнається стаціонарним лише тоді, коли обидва тести одночасно підтверджують стаціонарність (is_stationary = adf_stationary and kpss_stationary). Така логіка підвищує надійність висновку, оскільки тести мають різні нульові гіпотези і доповнюють один одного.

Крок 2. Ітеративне перетворення ряду до стаціонарного вигляду.

Логарифмічні перетворення є класом перетворень, а не окремим перетворенням, і в багатьох галузях науки логарифмічні нормальні змінні (тобто нормально розподілені після логарифмічного перетворення) є відносно поширеними. Логарифмічні нормальні змінні, здається, є більш поширеними, коли на результати впливає багато незалежних факторів (наприклад, біологічні результати), що також поширено в соціальних науках. Коротко кажучи, логарифм − це степінь (експонента), до якого потрібно піднести основне число, щоб отримати початкове число. Будь-яке задане число можна виразити як ух нескінченною кількістю способів. Наприклад, якщо ми говоримо про основну систему 10, 1 дорівнює 100, 100 дорівнює 102, 16 дорівнює 101,2 тощо. Таким чином, log10(100)=2 та log10(16)=1,2. Іншим поширеним варіантом є натуральний логарифм, де константа e (2,7182818…) є основою. У цьому випадку натуральний логарифм числа 100 дорівнює 4,605. Як показано на цьому прикладі, основа логарифма може бути майже будь-яким числом, що створює нескінченну кількість варіантів перетворення [14].

Аргумент про те, що слід враховувати різноманітні перетворення, сумісний з твердженням, що метод Бокса-Кокса може бути найкращою практикою перетворення даних [14].

Хоча це не реалізовано в усіх статистичних пакетах, існують способи оцінки лямбда, коефіцієнта перетворення Бокса-Кокса, за допомогою будь-якого статистичного пакета або вручну для автоматичної оцінки впливу вибраного діапазону λ [15]. Враховуючи, що λ може приймати майже нескінченну кількість значень, ми теоретично можемо калібрувати перетворення, щоб воно було максимально ефективним у переміщенні змінної до нормальності, незалежно від того, чи має вона негативну чи позитивну асиметрію [16].

Операція першої різниці має просте представлення через B. Припустимо, що y − перша різниця Y. Тоді для будь-якого t:

$$y_t=Y_t-Y_{(t-1)}=Y_t-BY_t=(1-B)\ Y_t \qquad (2.10)$$

Тепер, якщо z − перша різниця y, тобто z − друга різниця Y, тоді маємо:

$$z_t=y_t-y_{(t-1)}=(1-B)\ y_t=(1-B)((1-B)\ Y_t)=(1-B)^2\ Y_t \qquad (2.11)$$

Отже, друга різниця Y отримується множенням на коефіцієнт $(1-B)^2$, і загалом d-та різниця Y буде отримана множенням на коефіцієнт $(1-B)^d$.

Включення першої різниці еквівалентно множенню Y_t на коефіцієнт 1-B.

Кожен коефіцієнт 1-B, що з'являється в лівій частині рівняння, представляє порядок диференціації.

Якщо один з коренів дуже близький до 1 (так званий "одиничний корінь"), то часовий ряд не був належним чином стаціонаризований, і, ймовірно, було б краще використати ще одну несезонну різницю.

Якщо ряд нестаціонарний, послідовно застосовуються перетворення за схемою none → log → diff_1 → diff_2.

– логарифмування (np.log) стабілізує дисперсію (усуває гетероскедастичність), що характерно для даних про популярність з різким зростанням;

– перше різницювання (s.diff()) усуває лінійний тренд, перетворюючи рівні на приріст;

– друге різницювання застосовується при наявності квадратичного тренду.

Після кожного перетворення тест повторюється. Щойно стаціонарність досягнута – цикл зупиняється. Всі застосовані перетворення фіксуються у списку transformations і виводяться у звіті.

Декомпозиція часового ряду (метод _decompose). Для виявлення структури динаміки популярності застосовується метод STL-декомпозиції (Seasonal and Trend decomposition using Loess). Це сучасна альтернатива класичній адитивній/мультиплікативній декомпозиції, стійка до викидів завдяки параметру robust=True.

Сезонна декомпозиція – це статистичний метод розбиття часового ряду на його основні компоненти, які часто включають тренд, сезонні закономірності та залишкові (або помилкові) компоненти. Мета полягає в тому, щоб розділити різні джерела варіації в даних, щоб краще зрозуміти та проаналізувати кожен компонент окремо [17].

Локально зважене згладжування діаграми розсіювання або Лесс – це метод непараметричної регресії, який використовується для згладжування даних. У контексті аналізу часових рядів, сезонно-трендова декомпозиція з використанням Лесса (STL) – це специфічний метод декомпозиції, який використовує метод Лесса для розділення часового ряду на його трендові, сезонні та залишкові компоненти. STL особливо ефективний для обробки даних часових рядів зі складними та нелінійними закономірностями. У STL декомпозиція виконується шляхом ітеративного застосування згладжування Лесса до часових рядів. Цей

процес допомагає враховувати як короткострокові, так і довгострокові коливання даних, що робить його надійним методом декомпозиції часових рядів з нерегулярними або змінними закономірностями [17].

STL розкладає вихідний ряд на три компоненти [17]:

1. Тренд (Trend) – довгострокова динаміка популярності підкатегорії.
2. Сезонна складова (Seasonal) – циклічні коливання з автоматично визначеним або заданим вручну періодом.
3. Залишок (Residual) – випадкова складова, яку не пояснюють тренд і сезонність.

Сезонний період визначається автоматично методом _detect_period за частотою індексу: 7 для денних даних, 52 для тижневих, 12 для місячних, 4 для квартальних. Декомпозиція виконується на вихідному (нетрансформованому) ряді, що дозволяє інтерпретувати результати в оригінальних одиницях виміру.

Прогнозування популярності − SARIMA з перебором параметрів (метод _fit_and_forecast).

Прогнозування реалізовано на основі моделі SARIMA (Seasonal AutoRegressive Integrated Moving Average) − узагальнення ARIMA з явним урахуванням сезонності [18].

Модель сезонної авторегресивної інтегрованої ковзної середньої (SARIMA) виникає як складне розширення структури ARIMA, ретельно розроблене для вирішення складнощів, що виникають через сезонні закономірності. Моделі SARIMA особливо вигідні, коли дані демонструють регулярні періодичні коливання, які є поширеними в різних практичних застосуваннях, таких як щомісячні роздрібні продажі, квартальні економічні результати та щорічні кліматичні вимірювання. Основною передумовою моделей SARIMA є їхня здатність розкладати дані часових рядів на несезонні та сезонні компоненти, тим самим забезпечуючи більш нюансоване та точне представлення базового процесу генерації даних [18].

Модель SARIMA позначається як SARIMA(p,d,q)(P,D,Q,s), де параметри інкапсулюють як несезонні, так і сезонні аспекти даних. Тут 'p', 'd' та 'q'

відповідають порядку авторегресивної дії, ступеню несезонної диференціації та порядку ковзної середньої відповідно [18].

Структура моделі SARIMA(p,d,q)(P,D,Q,s) представлена в таблиці 2.3 [18]:

Таблиця 2.3 – Структура моделі SARIMA

Параметр	Значення	Опис
p	0–2	порядок авторегресії (AR)
d	0–1	порядок інтегрування (різницювання)
q	0–2	порядок ковзного середнього (MA)

Алгоритм підбору моделі: метод _build_candidate_orders формує всі комбінації параметрів у заданих діапазонах і сортує їх за сумарним порядком (від простих до складних). З них беруться перші 20 кандидатів (_MAX_MODELS = 20). Для кожного кандидата виконується підгонка моделі SARIMAX з бібліотеки statsmodels. Краща модель обирається за критерієм Акаіке (AIC) − мінімальне значення AIC відповідає оптимальному балансу між точністю і складністю моделі. Сезонна складова у SARIMA включається лише якщо довжина ряду не менша за два повних сезонних цикли.

Прогноз та довірчий інтервал: після вибору кращої моделі метод get_forecast(steps=n_periods) генерує точковий прогноз і 95% довірчий інтервал для заданої кількості майбутніх періодів. Результати повертаються як pd.Series з майбутнім часовим індексом. Всі результати аналізу записуються у три окремі таблиці бази даних: ts_stationarity_results (результати тестів стаціонарності), ts_decomposition_results (компоненти декомпозиції для кожного часового кроку) і ts_forecast_results (прогнозні значення з довірчими інтервалами). Після запису оновлюються матеріалізовані представлення SQL. Клас TimeSeriesVisualizer реалізує графічне відображення всіх етапів аналізу. Метод plot_full_report формує зведений семипанельний звіт, що включає вихідний ряд, rolling-статистики, таблицю результатів тестів стаціонарності, три графіки компонент декомпозиції, ACF/PACF корелограму та прогноз з довірчим інтервалом.

3 ДЕМОНСТРАЦІЯ, АНАЛІЗ ТА УЗАГАЛЬНЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Збір даних (Scraper)

Система складається з двох незалежних, але пов'язаних між собою модулів: модуля збору даних (main.py + parser.py) та модуля аналізу і візуалізації (run_analysis.py + analysis.py + visualizer.py). Обидва модулі використовують спільну базу даних PostgreSQL через асинхронний клієнт database.py.

main.py запускає веб-скрапер, який обходить сторінки товарів з певного сайту (URL задається через змінну середовища BASE_URL). Скрапер починає зі сторінки-лістингу, збирає перелік URL окремих товарів, потім послідовно відвідує кожен з них та зберігає отримані дані в базу. Кожен рядок показує статус (✓ − успішно, ✗ − помилка), кількість зібраних елементів на сторінці та, за наявності, деталі помилок JavaScript або мережевих збоїв. Усі зібрані товари зберігаються в таблицю products у PostgreSQL.

Приклад успішного скрапінгу зображений на рисунку 3.1.

```
2026-03-31 23:04:32,341 [INFO] database — Database pool created (min=2, max=10)
2026-03-31 23:04:33,807 [INFO] database — Database initialised from D:\Users\Vlad\OneDrive\Документ
2026-03-31 23:04:34,008 [INFO] database — Database pool created (min=2, max=10)
2026-03-31 23:04:34,844 [INFO] scraper — Fetching listing page 1: https://www.evans.co.uk/clothing
2026-03-31 23:04:49,243 [INFO] scraper — Page 1 — found 56 product URLs (total so far: 56)
2026-03-31 23:04:49,243 [INFO] scraper — Listing crawl done — 1 page(s), 56 product URL(s)
2026-03-31 23:05:56,949 [INFO] scraper — Scrape complete — 56/56 succeeded
[✓] https://www.evans.co.uk/evans-curve-chocolate-brown-co-ord-617274-p
    elements: 6
[✓] https://www.evans.co.uk/evans-curve-black-wide-leg-formal-trousers-616616-p
    elements: 12
[✓] https://www.evans.co.uk/evans-curve-navy-blue-wide-leg-linen-trousers-p
    elements: 12
[✓] https://www.evans.co.uk/evans-curve-chartreuse-green-cotton-broderie-blouse-p
    elements: 6
[✓] https://www.evans.co.uk/evans-curve-fit-blue-midwash-cropped-straight-leg-jeans-616540-p
    elements: 9
[✓] https://www.evans.co.uk/evans-curve-fit-indigo-straight-leg-jeans-p
    elements: 20
[✓] https://www.evans.co.uk/evans-2-pack-curve-black-stretch-leggings-p
    elements: 6
```

Рисунок 3.1 – Приклад успішного скрапінгу

Дані, які зібрав скрипт, продемонстровані на рисунку 3.2.

Results Messages public.products

gender	age_group	color	size	product_type	condition	availability	price	sale_price	rat
female	adult	Blue	16	Home>Plus Size Tops>Plus Size Floral Tops	new	out_of_stock	25.0 ZAR	NULL	NULL
female	adult	Blue	18	Home>Plus Size Tops>Plus Size Floral Tops	new	in_stock	25.0 ZAR	NULL	NULL
female	adult	Blue	20	Home>Plus Size Tops>Plus Size Floral Tops	new	in_stock	25.0 ZAR	NULL	NULL
female	adult	Blue	22-24	Home>Plus Size Tops>Plus Size Floral Tops	new	in_stock	25.0 ZAR	NULL	NULL
female	adult	Blue	26-28	Home>Plus Size Tops>Plus Size Floral Tops	new	in_stock	25.0 ZAR	NULL	NULL
female	adult	Blue	30-32	Home>Plus Size Tops>Plus Size Floral Tops	new	in_stock	25.0 ZAR	NULL	NULL
female	adult	Red	16	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Red	18	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Red	20	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Red	22-24	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Red	26-28	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Red	30-32	Home>Plus Size Shirts & Blouses>Plus Size Blouses	new	in_stock	28.0 ZAR	NULL	NULL
female	adult	Black	Short 16	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL
female	adult	Black	Short 18	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL
female	adult	Black	Short 20	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL
female	adult	Black	Short 22-24	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL
female	adult	Black	Short 26-28	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL
female	adult	Black	Short 30-32	Home>Plus Size Trousers>Plus Size Wide Leg Trousers	new	in_stock	35.0 ZAR	NULL	NULL

Рисунок 3.2 – Приклад даних, які зібрав скрапінг скрипт

3.2 Аналіз часових рядів і прогнозування

Запускається командою python run_analysis.py з різними параметрами. Це головний аналітичний інструмент.

Повний конвеєр – 6 кроків.

Крок 1. Побудова часових рядів популярності. Скрипт читає таблицю products, групує товари за підкатегорією та календарним днем і рахує кількість унікальних груп товарів (item_group_id) на кожен день. Результат записується в таблицю popularity_metrics. Таким чином, метрика "популярність" − це кількість унікальних продуктових груп, які з'явилися в певній підкатегорії за день.

Крок 2. Фільтрація підкатегорій. Завантажуються всі підкатегорії, що мають щонайменше N точок даних (за замовчуванням 14). Для кожної підкатегорії розраховуються: кількість точок, середнє, максимальне значення, дата початку і кінця ряду.

Крок 3. Визначення популярних підкатегорій. Підкатегорії ранжуються за середньою популярністю. За замовчуванням "популярними" вважаються ті, що знаходяться вище медіани (50-й перцентиль). Користувач може замість цього задати конкретну кількість --top N. У консоль виводяться дві таблиці −

"популярні" та "інші" − з рейтингом, кількістю точок, середнім і максимальним значенням.

Крок 4. Аналіз кожної популярної підкатегорії. Для кожної підкатегорії послідовно виконується повний аналітичний конвеєр.

Крок 5. Візуалізація. Для кожного результату генерується графік обраного типу, який зберігається на диск або показується інтерактивно.

Крок 6. Зведена таблиця прогнозів. У консоль виводиться порівняльна таблиця по всіх підкатегоріях: останнє спостережене значення, середнє прогнозоване значення та зміна у відсотках.

Скрипт для аналізу даних можна запустити з наступними параметрами, представленими в таблиці 3.1.

Таблиця 3.1 – Параметри командного рядка

Параметр	За замовчуванням	Що контролює
--skip-populate	вимк.	Пропустити оновлення popularity_metrics (дані вже є)
--top N	авто (50-й перц.)	Аналізувати лише топ-N підкатегорій
--min-points N	14	Мінімальна довжина ряду для включення в аналіз
--forecast-periods N	30	Горизонт прогнозу (кількість майбутніх точок)
--seasonal-period N	авто	Примусово задати сезонний період (7, 12, 52…)
--output-dir PATH	reports/	Папка для збереження графіків
--no-save	вимк.	Показувати графіки інтерактивно замість збереження
--stats	вимк.	Друкувати повну статистику в консоль
--plot TYPE	full	Тип графіку: full, series, stationarity, decomposition, acf-pacf, forecast

У консолі − структурований кольоровий звіт по кожній підкатегорії: довжина ряду та часовий діапазон, застосовані перетворення, результати тестів стаціонарності (ADF p-value, KPSS p-value, вердикт), параметри обраної SARIMA-моделі та її AIC, рекомендація щодо подальшої обробки. При --stats додатково виводяться повні дескриптивні статистики кожного компонента декомпозиції (середнє, стд., мін., макс.) та поточнева таблиця прогнозу з довірчими інтервалами.

На диску (reports/) − PNG-файли графіків. Залежно від обраного типу --plot це може бути один з шести варіантів. При виборі full генерується семипанельний зведений рисунок: сирий ряд з трансформованою версією; ковзні середні (7 і 30 періодів) з коридором ±σ; таблиця результатів тестів стаціонарності; три панелі декомпозиції (тренд, сезонність, залишок); корелограма ACF; прогноз з 95% довірчим інтервалом та вертикальним роздільником між історичними та прогнозними значеннями.

У базі даних − три таблиці з результатами (ts_stationarity_results, ts_decomposition_results, ts_forecast_results) та оновлені матеріалізовані уявлення (materialized views), що автоматично оновлюються після завершення аналізу.

Варто зазначити, що для отримання результатів та прогнозів за останні 6 років, дані по категоріях одягу були штучно згенеровані випадковим чином.

На рисунку 3.3 зображено зведену статистику для підкатегорії Plus Size Straight Leg Jeans за метрикою product_count.

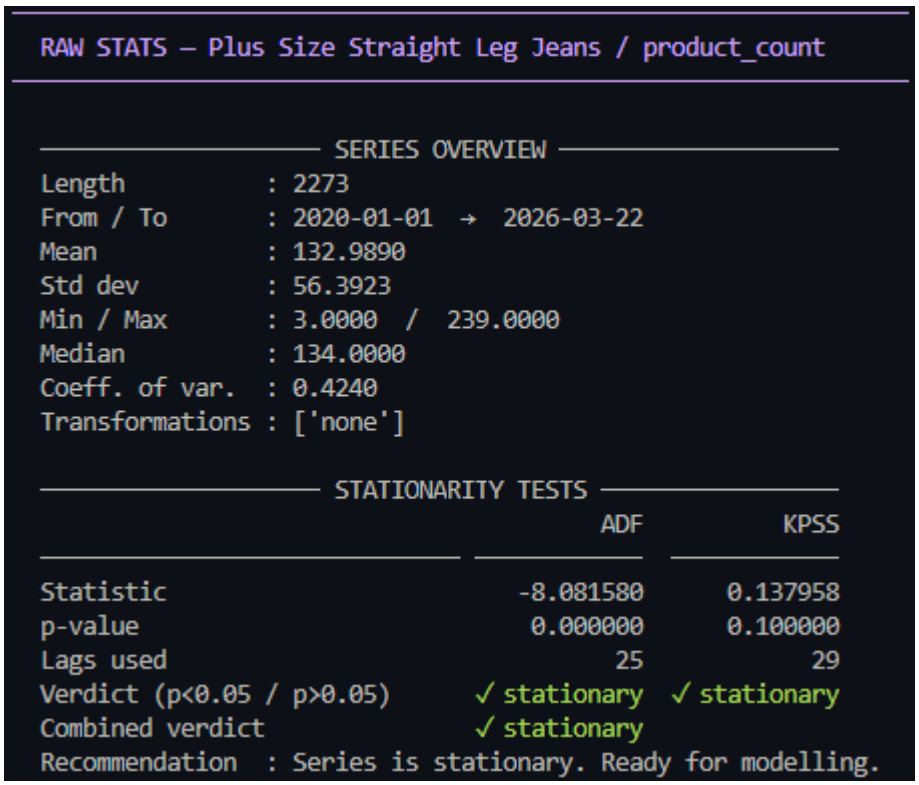
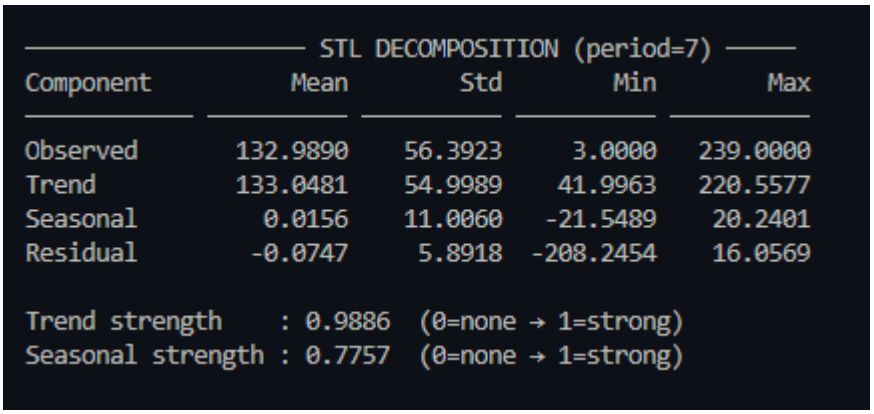


Рисунок 3.3 – Зведена статистика для підкатегорії Plus Size Straight Leg Jeans за метрикою product_count

Огляд ряду показує, що дані охоплюють період з 1 січня 2020 по 22 березня 2026 − загалом 2273 спостережень (щоденні точки за понад 6 років). Середня кількість унікальних товарних груп у цій підкатегорії на день становить близько 133, при стандартному відхиленні 56 − тобто значення коливаються досить широко. Мінімум становить 3, максимум − 239 товарів за день, медіана близька до середнього (134), що свідчить про симетричний розподіл. Коефіцієнт варіації 0.424 означає помірну відносну мінливість (42%). Жодних трансформацій до ряду застосовано не було − він одразу виявився придатним для моделювання.

Тести стаціонарності підтверджують це: обидва тести (ADF і KPSS) дають однозначний висновок − ряд є стаціонарним. ADF-статистика −8.08 з p-value ≈ 0 говорить про відсутність одиничного кореня (тобто тренд не домінує). KPSS p-value = 0.1 (> 0.05) означає, що гіпотеза про стаціонарність не відхиляється. Рекомендація системи: "Series is stationary. Ready for modelling" − ряд готовий до SARIMA-моделювання без додаткової обробки.

Результати розкладання ряду Plus Size Straight Leg Jeans методом STL з тижневим сезонним періодом (7 днів) продемонстровані на рисунку 3.4.



STL DECOMPOSITION (period=7)

Component	Mean	Std	Min	Max
Observed	132.9890	56.3923	3.0000	239.0000
Trend	133.0481	54.9989	41.9963	220.5577
Seasonal	0.0156	11.0060	-21.5489	20.2401
Residual	-0.0747	5.8918	-208.2454	16.0569

Trend strength : 0.9886 (0=none → 1=strong)
Seasonal strength : 0.7757 (0=none → 1=strong)

Рисунок 3.4 – Розкладання ряду Plus Size Straight Leg Jeans методом STL з тижневим сезонним періодом (7 днів)

Таблиця показує описову статистику кожного з чотирьох компонентів. Observed − оригінальний ряд, середнє 133, розкид від 3 до 239. Trend (тренд) − має практично те саме середнє (133), але значно менший розкид (41–220), бо позбавлений короткострокових коливань: це "очищена" довгострокова динаміка попиту. Seasonal (сезонна складова) − середнє майже нульове (0.016), коливається між −21.5 та +20.2, що означає тижневий патерн амплітудою до ±20 товарів відносно тренду (наприклад, більше товарів у будні, менше у вихідні або навпаки). Residual (залишок) − теж близький до нуля в середньому (−0.075), але має викиди до ±208, що свідчить про окремі аномальні дні в даних.

Найважливіші два показники внизу: сила тренду 0.9886 (максимально близька до 1) означає, що тренд пояснює майже весь ряд − динаміка підкатегорії має виражену довгострокову спрямованість. Сила сезонності 0.7757 − висока, але не домінуюча, тобто тижневий цикл помітний і статистично значущий, але тренд є важливішим чинником.

Зведена таблиця прогнозів по всіх 12 проаналізованих популярних підкатегоріях продемонстрована на рисунку 3.5.

FORECAST SUMMARY — predicted mean vs last observed value

Subcategory	Last observed	Forecast mean	Change
Plus Size Blouses	12.0	335.5	↑ 323.5
Plus Size Jackets	2.0	315.6	↑ 313.6
Plus Size Straight Leg Jeans	3.0	177.2	↑ 174.2
Plus Size Floral Tops	1.0	0.2	↓ 0.8
Plus Size Linen Trousers	2.0	0.2	↓ 1.8
Plus Size Animal Print Tops	1.0	0.1	↓ 0.9
Plus Size Vests & Camis	2.0	0.1	↓ 1.9
Plus Size Short Sleeve Tops	3.0	0.1	↓ 2.9
Plus Size Casual Tops	3.0	0.1	↓ 2.9
Plus Size Maxi Dresses	5.0	0.1	↓ 4.9
Plus Size Wide Leg Trousers	10.0	0.1	↓ 9.9
Plus Size T-Shirts	3.0	-0.2	↓ 3.2

2026-03-22 14:43:33,749 [INFO] database — Database pool closed

Рисунок 3.5 – Зведена таблиця прогнозів по всіх 12 проаналізованих популярних підкатегоріях

Це зведена таблиця прогнозів по всіх 12 проаналізованих популярних підкатегоріях, де порівнюється останнє спостережене значення (станом на 22.03.2026) з середнім прогнозованим значенням на наступні 30 днів.

Результати поділяються на дві групи. Підкатегорії зі зростанням (зелений колір, стрілка вгору) − три позиції, де модель прогнозує суттєве зростання кількості товарних груп відносно останнього дня. Plus Size Blouses: останнє значення 12, прогноз 335.5 − зростання на 323.5. Plus Size Jackets: 2 → 315.6 (+313.6). Plus Size Straight Leg Jeans: 3 → 177.2 (+174.2). Такі великі стрибки пояснюються тим, що останній день ряду припав на нетиповий мінімум, тоді як модель орієнтується на середній рівень ряду.

Підкатегорії зі спаданням (червоний колір, стрілка вниз) − решта 9 підкатегорій, де прогнозовані середні значення нижчі за останнє спостереження. Наприклад, Plus Size Wide Leg Trousers: 10 → 0.1 (−9.9), Plus Size Maxi Dresses: 5 → 0.1 (−4.9). Низькі прогнозовані значення (0.1–0.2) для більшості підкатегорій

можуть свідчити про те, що ці підкатегорії мають нерегулярну або затухаючу динаміку, і модель прогнозує їх повернення до дуже низького базового рівня.

Last observed (останнє значення) для багатьох категорій дуже низьке (1.0, 2.0, 3.0), а Forecast mean − високе (300+). Це сталося тому, що ми згенерували 6 років "історії" з високими показниками, а наш реальний останній день який ми зіскраплили мав лише кілька товарів.

Система виявила аномальне падіння в останній день порівняно з багаторічною історією, але прогнозує "повернення до норми" на основі історичних даних.

Порівняємо також дві згенеровані статистики двох різних підкатегорій – однієї з спадаючим трендом, а однієї зі зростаючим. На рисунках 3.6 та 3.7 продемонтсрована статистика для категорії Plus Size Wide Leg Trousers.

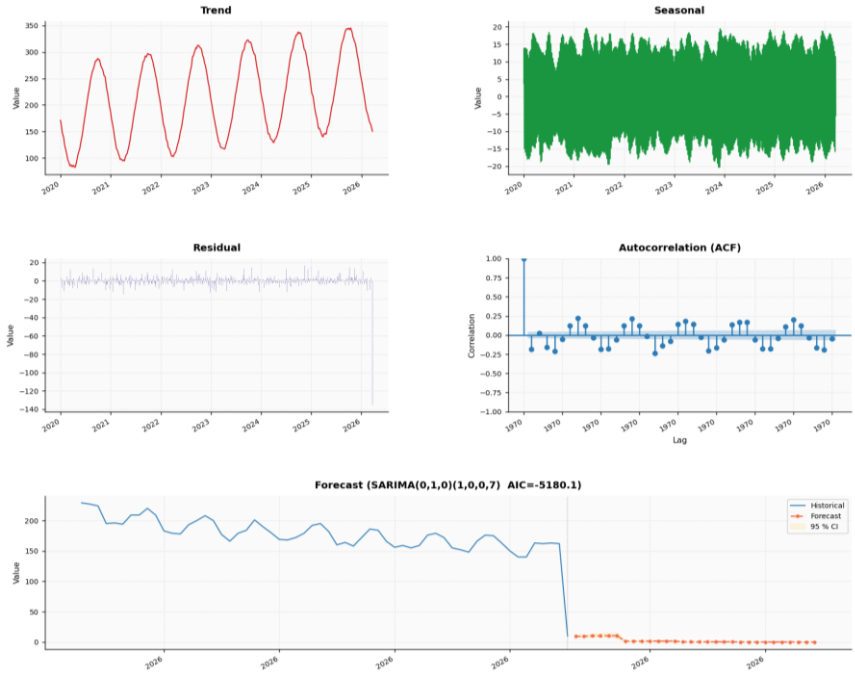


Рисунок 3.6 – Графіки для категорії Plus Size Wide Leg Trousers

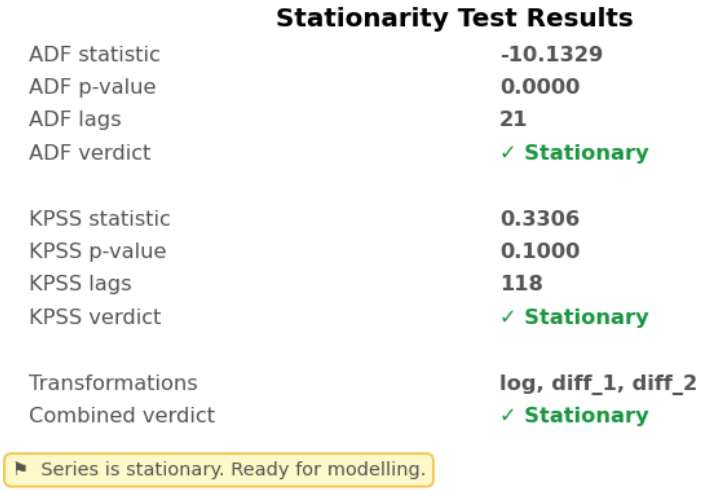


Рисунок 3.7 – Статистика для категорії Plus Size Wide Leg Trousers

Чому тут прогноз падає до нуля? Трансформації: log + diff_1. Це ключовий чинник. Щоб досягти стаціонарності, система була змушена застосувати логарифмування і потім перше диференціювання. Після цих перетворень модель працює не з абсолютними значеннями, а з відносними щоденними змінами у логарифмічному масштабі.

Коли в останній день значення впало з ~290 до ~0, логарифмічне диференціювання перетворило це на екстремально від'ємний "шок" ($\log(0) \to -\infty$). Обрана модель SARIMA(0,0,1)(1,0,0,7) містить лише один член ковзного середнього (MA=1), тобто вона дуже чутлива до останньої помилки прогнозу − і цей шок стає домінуючим сигналом для всього прогнозного горизонту.

ACF: корелограма майже пуста. На графіку ACF після lag=0 значення одразу падають майже до нуля і залишаються в межах довірчого інтервалу. Це означає, що після трансформації ряд поводиться як білий шум − немає ні тренду, ні сезонної пам'яті, яка могла б "витягнути" прогноз угору. Модель не бачить жодної структури, яка б підштовхувала прогноз до середнього рівня. Тренд: завершується спаданням. Щоб уникнути таких аномалій, в майбутньому модель можна покращи, дати їй наприклад інформацію про те, чи повноцінно завершився скрапінг сайту, чи можливо він завершився з помилками, і тому дані за цей останній день можна не враховувати.

На рисунках 3.8 та 3.9 продемонтсрована статистика для категорії Plus Size Blouses.

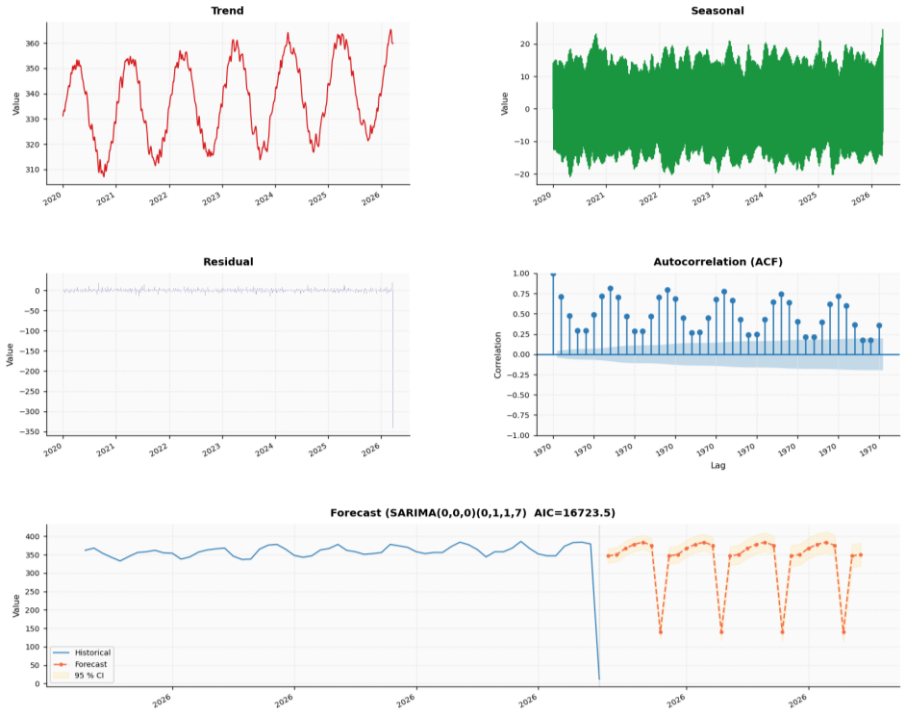


Рисунок 3.8 – Графіки для категорії Plus Size Blouses

Stationarity Test Results

ADF statistic	**-3.0847**
ADF p-value	**0.0277**
ADF lags	**24**
ADF verdict	✓ **Stationary**
KPSS statistic	**0.4595**
KPSS p-value	**0.0515**
KPSS lags	**25**
KPSS verdict	✓ **Stationary**
Transformations	**none**
Combined verdict	✓ **Stationary**

⚑ Series is stationary. Ready for modelling.

Рисунок 3.9 – Статистика для категорії Plus Size Blouses

Чому прогноз цієї категорії не впав до нуля? Трансформації: none Ряд одразу виявився стаціонарним без жодних перетворень. Модель SARIMA(0,0,0)(0,1,1,7) працює безпосередньо з оригінальними значеннями. Параметр сезонного диференціювання D=1 означає, що модель прогнозує зміни

відносно того самого дня тижня попереднього тижня − вона "дивиться назад" на 7 днів, а не на 1. Останній аномальний день впливає лише на одну точку в сезонному диференціюванні, а не на весь ряд.

ACF: яскраво виражена тижнева сезонність На графіку ACF чітко видно регулярні піки на кожному 7-му лагу − значення автокореляції значно виходять за межі довірчого інтервалу і зберігаються протягом багатьох тижнів. Це свідчить про те, що ряд має сильну та стійку тижневу пам'ять: значення понеділка корелює з наступним понеділком, вівторка − з вівторком і так далі. Модель вловлює цю структуру і продовжує її у майбутнє.

Тренд: зростає на кінці. На відміну від Wide Leg Trousers, тренд Blouses на кінець 2025–2026 іде вгору, що формує позитивний базовий рівень для прогнозу. Модель "пам'ятає" тижневий цикл попередніх тижнів і відтворює хвилеподібний патерн на прогнозному горизонті, повністю ігноруючи аномальний останній день як одиничний сезонний викид.

Приклад тих самих прогнозів, без останнього аномального дня, продемонстрований на рисунках 3.10 та 3.11.

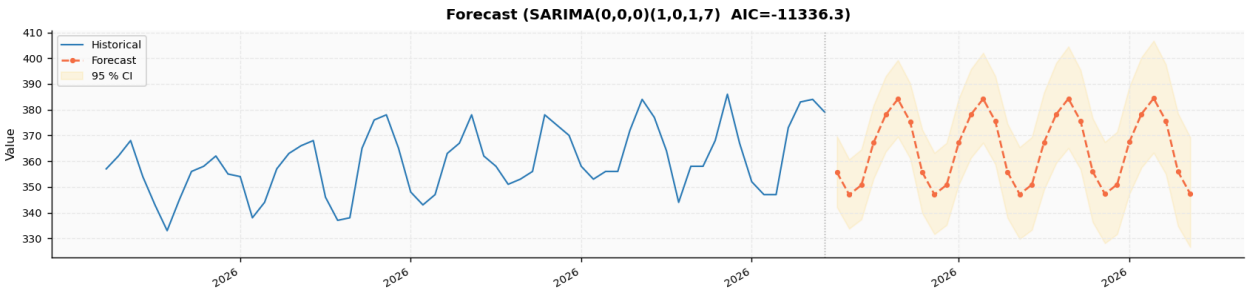


Рисунок 3.10 – Прогнозування кількості товарів категорії Plus Size Blouses без аномального останнього дня

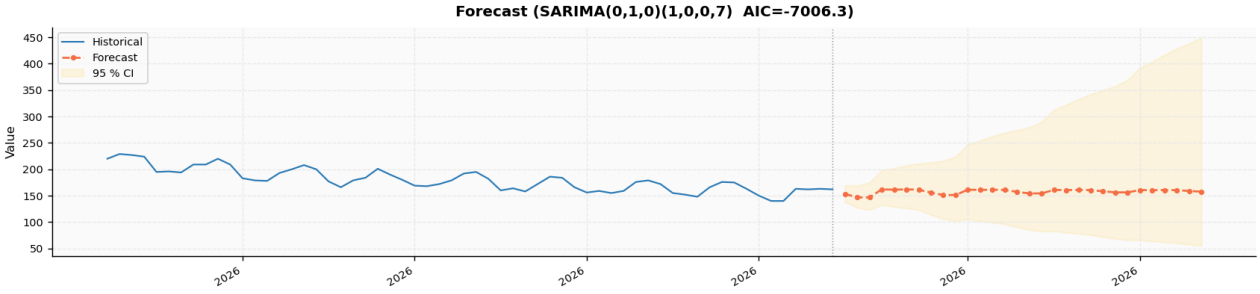


Рисунок 3.11 – Прогнозування кількості товарів категорії Plus Size Wide Leg Trousers без аномального останнього дня

Plus Size Blouses − SARIMA(0,0,0)(1,0,1,7), AIC = −11336.3. Прогноз демонструє високу якість. Модель впевнено відтворює тижневий сезонний патерн, амплітуда і частота коливань прогнозу (350–385) відповідають останнім спостереженням в історичних даних. Довірчий інтервал 95% залишається відносно вузьким і рівномірно розширюється з часом, що свідчить про стабільну невизначеність. Від'ємне AIC з великим модулем підтверджує хорошу відносну якість підбору моделі. Прогноз є практично придатним.

Plus Size Wide Leg Trousers − SARIMA(0,1,0)(1,0,0,7), AIC = −7006.3. Прогноз значно слабший. Модель фіксує середній рівень (~160–170 товарів) і майже повністю втрачає сезонну хвилю, яка чітко присутня в історичних даних. Найбільша проблема − катастрофічно широкий довірчий інтервал, який до кінця горизонту розтягується від ~50 до ~400, фактично охоплюючи весь діапазон можливих значень. Це означає, що модель не дає корисної інформації про майбутнє − лише констатує середній рівень із надзвичайно високою невизначеністю. Причина − агресивне диференціювання (d=1) на ряді з високою волатильністю, що "стирає" сезонну структуру.

SARIMA grid search дає прийнятні результати для стаціонарних або помірно нестаціонарних рядів (Blouses), але для рядів із сильним трендом і одночасно високою волатильністю (Wide Leg Trousers) якість прогнозу суттєво знижується.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

4.1.1 Аналіз небезпечних та шкідливих факторів під час розробки й експлуатації системи скрапінгу онлайн-магазинів одягу

Розлади опорно-рухового апарату (РОМ) впливають на м'язи, нерви, кровоносні судини, зв'язки та сухожилля. Працівники багатьох різних галузей та професій можуть піддаватися впливу факторів ризику на роботі, таких як підняття важких предметів, нахиляння, тягнення рук над головою, штовхання та перетягування важких вантажів, робота в незручних позах тіла та повторюване виконання тих самих або подібних завдань. Вплив цих відомих факторів ризику розвитку РОМ підвищує ризик травмування працівника [19].

Розлади опорно-рухового апарату у програмістів виникають внаслідок тривалої роботи за комп’ютером та впливають на м’язи, суглоби, зв’язки, сухожилля і нервову систему. Під час розробки програмного забезпечення, зокрема системи скрапінгу онлайн-магазинів одягу з функціями аналізу та прогнозування, програміст тривалий час перебуває у сидячому положенні, постійно працює з клавіатурою та мишею, а також виконує повторювані рухи руками. Це може призводити до болю в шиї, спині, плечах, кистях рук та розвитку професійних захворювань, таких як тунельний синдром або остеохондроз [19].

РОМ, пов’язані з роботою програміста, можна значною мірою попередити завдяки правильній організації робочого місця та дотриманню ергономічних вимог. Ергономіка − це адаптація умов праці до фізіологічних особливостей людини. Вона допомагає зменшити м’язову втому, підвищити продуктивність праці та знизити ризик виникнення професійних захворювань [19].

Розлади опорно-рухового апарату є однією з найбільш поширених причин погіршення самопочуття працівників ІТ-сфери. Тривала робота за комп’ютером без достатніх перерв може викликати хронічну втому, біль у попереку, порушення

постави та зниження працездатності. У важких випадках це може призводити до тимчасової втрати працездатності та необхідності тривалого лікування [19].

Необхідно регулярно оцінювати ефективність впроваджених заходів з охорони праці. Це дозволяє своєчасно виявляти недоліки та вдосконалювати умови праці програмістів [19].

4.1.2 Організація безпечного робочого місця програміста при розробці системи аналізу та прогнозування даних онлайн-магазинів

Правильне розміщення компонентів та аксесуарів для настільної комп’ютерної робочої станції дозволить вам працювати в нейтральному положенні тіла, допоможе вам працювати ефективніше, а також працювати комфортніше та безпечніше [20].

Робоча станція з ноутбуком створює особливі труднощі через її конструкцію, розмір та різноманітність областей, у яких вона використовується. Хоча багато аспектів цього електронного інструменту будуть застосовні до ноутбуків, при роботі з ноутбуками можуть знадобитися спеціальні міркування.

Добре спроектований та належним чином налаштований стілець є важливим елементом безпечного та продуктивного комп’ютерного робочого місця. Гарний стілець забезпечує необхідну підтримку спини, ніг, сідниць та рук, одночасно зменшуючи вплив незручних поз, контактного стресу та надмірних зусиль.

Вибір відповідного монітора та його розміщення у відповідному положенні допомагає зменшити вплив надмірних навантажень, незручних поз та відблисків зверху. Це допомагає запобігти можливим наслідкам для здоров'я, таким як надмірна втома, напруга очей, біль у шиї та спині. Сядьте на зручній відстані від монітора, щоб ви могли легко читати весь текст, тримаючи голову та тулуб у вертикальному положенні, а спину спираючись на стілець. Зазвичай, бажана відстань для перегляду становить від 50 до 100 см (від 20 до 40 дюймів) від ока до передньої поверхні екрана комп’ютера [21].

Добре спроектований та належним чином відрегульований стіл забезпечить достатній простір для ваших ніг, дозволить правильно розмістити компоненти та аксесуари комп'ютера, а також мінімізувати незручні пози та навантаження.

Правильний вибір та розташування клавіатури комп’ютера допомагає зменшити вплив незручних поз, повторення та контактного стресу. Клавіатури, вказівні пристрої або робочі поверхні, розташовані занадто високо або занадто низько, можуть призвести до незручного положення зап'ясть, рук і плечей. Наприклад, коли клавіатури розташовані занадто низько, ви можете друкувати, зігнувши зап'ястя, а коли клавіатури розташовані занадто високо, вам може знадобитися підняти плечі, щоб підняти руки. Виконання завдань з натисканням клавіш у таких незручних позах може призвести до дискомфорту в руках, зап'ястях і плечах [22].

Окрім звичайної миші, існують трекболи, сенсорні панелі, джойстики для пальців та шайби, і це лише деякі з них. Вибір та розташування вказівника/миші є важливим фактором у створенні безпечної комп’ютерної робочої станції. Якщо вказівник/миша не знаходиться поруч із клавіатурою, ви можете бути схильні до незручних поз, контактного навантаження або сильних напружень рук під час використання пристрою. Робота в такому положенні протягом тривалого часу створює навантаження на плече та руку і збільшує ймовірність того, що ви приймете незручне положення зап'ястя та плеча, що може призвести до захворювань опорно-рухового апарату [23].

4.1.3 Висновок до розділу "Охорона праці"

У результаті проведеного аналізу було визначено основні небезпечні та шкідливі фактори, що можуть виникати під час розробки та експлуатації системи скрапінгу онлайн-магазинів одягу.

Основну небезпеку для програміста становлять розлади опорно-рухового апарату, перевтома органів зору та статичне навантаження, пов’язані з тривалою роботою за комп’ютером.

Було встановлено, що дотримання ергономічних вимог та правильна організація робочого місця дозволяють суттєво знизити ризик виникнення професійних захворювань і підвищити комфорт та продуктивність праці. Важливими складовими безпечної роботи є використання ергономічного крісла та столу, правильне розташування монітора, клавіатури й миші, а також дотримання режиму праці та відпочинку.

Отже, забезпечення безпечних умов праці програміста є важливим елементом ефективної та безпечної розробки системи аналізу та прогнозування даних онлайн-магазинів одягу.

4.2 Безпека в надзвичайних ситуаціях

4.2.1 Забезпечення безпеки роботи з комп’ютерною технікою в умовах надзвичайних ситуацій

Надзвичайна ситуація на робочому місці – це ситуація, яка загрожує працівникам, клієнтам або населенню; порушує або зупиняє роботу; або завдає фізичної чи екологічної шкоди. Надзвичайні ситуації можуть бути природними або техногенними, і можуть включати урагани, торнадо, землетруси, повені, лісові пожежі, зимову погоду, розливи або викиди хімічних речовин, спалахи захворювань, викиди біологічних агентів, вибухи, пов'язані з ядерними або радіологічними джерелами, та багато інших небезпек. Багато видів надзвичайних ситуацій можна передбачити в процесі планування, що може допомогти роботодавцям і працівникам планувати інші непередбачувані ситуації [24].

Дії працівників у разі ураження електричним струмом [25]:

– терміново звільнити потерпілого від дії електричного струму (через відключення електроживлення в кімнаті, загального електроживлення на розподільному щиті або іншим способом);

– викликати швидку медичну допомогу (подзвонивши за міським телефоном 03);

– надати першу медичну допомогу потерпілому;

– якщо потерпілий знепритомнів, але дихає, його необхідно рівно і зручно вкласти, розстебнути одяг, створити приплив свіжого повітря і забезпечити повний спокій;

– при відсутності ознак життя до прибуття лікарів потерпілому необхідно робити штучне дихання.

Дії працівників у разі виникнення пожежі [25]:

– про виникнення пожежі в приміщеннях Міннауки негайно повідомити пожежну охорону за міським телефоном 01;

– при цьому необхідно назвати адресу Міннауки, зазначити кількість поверхів будівлі, місце виникнення пожежі, обстановку на пожежі, наявність людей, а також повідомити своє прізвище;

– вжити (по можливості) заходи на евакуацію людей, гасіння (локалізацію) пожежі з використанням первинних засобів пожежогасіння та на збереження матеріальних цінностей;

– повідомити про виникнення пожежі Міністра (заступників Міністра) чи відповідальну компетентну посадову особу та (або) чергового охорони;

– у разі необхідності, викликати інші аварійно-рятувальні служби (медичну, газорятувальну тощо).

4.2.2 Висновок до розділу "Безпека в надзвичайних ситуаціях"

У даному підрозділі розглянуто особливості забезпечення безпеки роботи з комп’ютерною технікою в умовах надзвичайних ситуацій. Встановлено, що надзвичайні ситуації можуть мати як природний, так і техногенний характер та становлять серйозну загрозу для життя і здоров’я працівників.

Проаналізовано основні дії працівників у разі виникнення найбільш поширених небезпечних ситуацій, зокрема ураження електричним струмом та пожежі. Підкреслено, що ключовими заходами безпеки є своєчасне відключення

електроживлення, оперативне повідомлення екстрених служб, надання першої допомоги постраждалим та організація евакуації людей із небезпечної зони.

Отже, дотримання чітко визначених алгоритмів дій у надзвичайних ситуаціях дозволяє мінімізувати ризики для працівників, зменшити можливі наслідки аварій та забезпечити безпечну експлуатацію комп’ютерної техніки навіть в умовах виникнення небезпечних подій.

ВИСНОВКИ

В першому розділі кваліфікаційної роботи:

– подано огляд предметної області електронної комерції, зокрема сегменту онлайн-магазинів одягу, та визначено особливості структури їх даних;

– розглянуто етичні та правові аспекти використання веб-скрапінгу у маркетингових дослідженнях;

– висвітлено теоретичні основи веб-скрапінгу та підходи до його застосування для збору даних з веб-ресурсів;

– проаналізовано правові обмеження та етичні принципи використання даних, отриманих з інтернет-магазинів;

– досліджено сучасні методи визначення популярних товарних підкатегорій та підходи до прогнозування їх популярності;

– обґрунтовано актуальність обраного напряму дослідження в умовах зростання обсягів даних електронної комерції;

– сформовано теоретичне підґрунтя для подальшого проєктування системи веб-скрапінгу та аналізу даних.

В другому розділі кваліфікаційної роботи:

– описано вибір мови програмування python як основного інструменту реалізації системи з урахуванням її можливостей для обробки даних та веб-скрапінгу;

– досліджено застосування об'єктно-орієнтованих технологій і принципів при проєктуванні програмної системи;

– подано порівняльний опис підходів до організації логіки аналізу отриманих результатів та обробки зібраних даних.

В третьому розділі кваліфікаційної роботи:

– розроблено програмну систему збору даних (scraper) з онлайн-магазинів одягу;

– запропоновано підхід до аналізу часових рядів для визначення популярних підкатегорій товарів;

– спроєктовано модуль прогнозування популярності підкатегорій на основі зібраних даних;

– протестовано роботу системи та проаналізовано отримані результати, що підтверджують ефективність запропонованого підходу.

В четвертому розділі кваліфікаційної роботи:

– розглянуто питання охорони праці під час розробки та експлуатації програмної системи веб-скрапінгу та аналізу даних;

– проаналізовано небезпечні та шкідливі фактори, що впливають на програміста під час роботи за комп’ютером;

– визначено вимоги до організації безпечного робочого місця програміста;

– описано правила безпеки роботи з комп’ютерною технікою в умовах надзвичайних ситуацій, включаючи дії при пожежі та ураженні електричним струмом.

Враховуючи усе вищеперераховане, можна зробити висновок, що всі поставлені завдання кваліфікаційної роботи були виконані в повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ

1. Khder, Moaiad Ahmad. "Web scraping or web crawling: State of art, techniques, approaches and application." International Journal of Advances in Soft Computing & Its Applications 13.3 (2021).

2. https://www.malwarebytes.com/blog/news/2024/03/facebook-spied-on-snapchat-users-to-get-analytics-about-the-competition

3. What is Web Scraping and How to Use It?. Geeksforgeeks. Blogs. 15.07.2025. URL: https://www.geeksforgeeks.org/blogs/what-is-web-scraping-and-how-to-use-it/ (дата звернення: 28.10.2025).

4. Is Web Scraping Legal? Covering All Aspects. Medium. 17.07.2024. URL: https://medium.com/@datajournal/is-web-scraping-legal-0df27c2e2ec6 (дата звернення: 28.10.2025).

5. Pawan G. Time Series Analysis and Forecasting. GeeksforGeeks. Machine learning. 19.12.2025. URL: https://www.geeksforgeeks.org/machine-learning/time-series-analysis-and-forecasting/ (дата звернення: 11.03.2026).

6. Знайомство з Google Трендами. Google Trends. URL: https://trends.google.com/explore.

7. TIOBE Index for March 2026. TIOBE. URL: https://www.tiobe.com/tiobe-index/.

8. Goldwasser M. H., Letscher D. Object-Oriented Programming in Python. Pearson Education Inc, 2014. 684 с.

9. Main principles of OOP. EPAM Campus. 13.01.2025. URL: https://campus.epam.com/en/blog/275.

10. Madasu V. K., Venna T. V. S. N. SOLID Principles in Software Architecture and Introduction to RESM Concept in OOP. Journal of Multidisciplinary Engineering Science and Technology. 2015. Т. 2, вип. 2. С. 3. ISSN 3159-0040.

11. Ramachandrappa N. C. SOLID Design Principles in Software Engineering. International Journal of Computer Trends and Technology. 2024. Т. 72, вип. 9. С. 18–23. ISSN 22312803. URL: https://doi.org/10.14445/22312803/IJCTT-V72I9P104.

12. Mushtaq R. TESTING TIME SERIES DATA FOR STATIONARITY / Université Paris. Paris, 2011. 19 c. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1911068.

13. Osborne J. W. Improving your data transformations: Applying the Box-Cox transformation. Practical Assessment, Research & Evaluation. 2010. Т. 15, вип. 12. С. 9. ISSN 1531-7714.

14. Box G. E. P., Cox D. R. An analysis of transformations. Journal of the Royal Statistical Society. 1964. Т. 26. С. 211–234.

15. Cleveland W. S. Full scale breaks, dot charts, and multibased logging. The American Statistician. 1984. Т. 38, вип. 4. С. 270–280.

16. Nau R. The mathematical structure of ARIMA models. Fuqua School of Business, Duke University. С. 8.

17. Bhakta S. S. Seasonal Decomposition of Time Series by Loess (STL). Geeksforgeeks. 23.07.2025. URL: https://www.geeksforgeeks.org/data-analysis/seasonal-decomposition-of-time-series-by-loess-stl/.

18. Majka M. Seasonal Time Series Analysis: Why SARIMA Outshines ARIMA. ResearchGate. 2024. С. 10. URL: https://www.researchgate.net/publication/384196885_Seasonal_Time_Series_Analysis_Why_SARIMA_Outshines_ARIMA.

19. Ergonomics. Occupational Safety and Health Administration. Safety and Health Topics. URL: https://www.osha.gov/ergonomics.

20. Workstation Components. Occupational Safety and Health Administration. Computer Workstations : Workstation Components. URL: https://www.osha.gov/etools/computer-workstations/components/.

21. Monitors. Occupational Safety and Health Administration. Computer Workstations : Workstation Components - Monitors. URL: https://www.osha.gov/etools/computer-workstations/components/monitors.

22. Keyboards. Occupational Safety and Health Administration. Computer Workstations : Workstation Components - Keyboards. URL: https://www.osha.gov/etools/computer-workstations/components/keyboards.

23. Pointer/Mouse. Occupational Safety and Health Administration. Computer Workstations : Workstation Components - Pointer/Mouse. URL: https://www.osha.gov/etools/computer-workstations/components/pointer-mouse.

24. Emergency Preparedness and Response: Getting Started. Occupational Safety and Health Administration. Emergency Preparedness and Response. URL: https://www.osha.gov/emergency-preparedness/getting-started.

25. Про охорону праці : НАКАЗ. МІНІСТЕРСТВО УКРАЇНИ У СПРАВАХ НАУКИ І ТЕХНОЛОГІЙ. від 11.03.1997, № 62. URL: https://zakon.rada.gov.ua/rada/show/v0062206-97#Text.

ДОДАТКИ

Додаток А

Тези конференцій

ЗБІРНИК НАУКОВИХ ПРАЦЬ

З МАТЕРІАЛАМИ VI МІЖНАРОДНОЇ НАУКОВОЇ КОНФЕРЕНЦІЇ

27 ЛЮТОГО 2026 РІК

М. ЧЕРКАСИ, УКРАЇНА

**«ІНТЕЛЕКТУАЛЬНИЙ РЕСУРС СЬОГОДЕННЯ:
НАУКОВІ ЗАДАЧІ, РОЗВИТОК ТА ЗАПИТАННЯ»**

Рисунок А.1 – Титульна сторінка конференції №1

ЗБІРНИК НАУКОВИХ
ПРАЦЬ З МАТЕРІАЛАМИ
VI МІЖНАРОДНОЇ
НАУКОВОЇ КОНФЕРЕНЦІЇ

ІНТЕЛЕКТУАЛЬНИЙ РЕСУРС СЬОГОДЕННЯ: НАУКОВІ ЗАДАЧІ, РОЗВИТОК ТА ЗАПИТАННЯ

27 лютого 2026 рік
м. Черкаси, Україна

Вінниця, Україна
«UKRLOGOS Group»
2026

Рисунок А.2 – Форзац з конференції №1

УДК 082:001
І-66

Організація, від імені якої випущено видання:
ГО «Міжнародний центр наукових досліджень»
Номер запису організації в Єдиному реєстрі громадських об'єднань: 1499141.

Голова оргкомітету: Сотник С.Г.

Верстка: Бабич Ю.В.
Дизайн: Бондаренко І.В.

Рекомендовано до видання Вченою Радою Інституту науково-технічної інтеграції та співпраці. Протокол № 7 від 26.02.2026 року.

*Конференцію зареєстровано Державною науковою установою у сфері управління Міністерства освіти і науки «Український інститут науково-технічної експертизи та інформації» в базі даних науково-технічних заходів України на поточний рік та бюлетені «План проведення наукових, науково-технічних заходів в Україні» (**Посвідчення № 517 від 10.06.2025**).*

*Збірник наукових праць з матеріалами конференції видано офіційно суб'єктом видавничої справи зі **Свідоцтвом ДК № 7860 від 22.06.2023**.*

І-66 **Інтелектуальний ресурс сьогодення: наукові задачі, розвиток та запитання**: збірник наукових праць з матеріалами VI Міжнародної наукової конференції, м. Черкаси, 27 лютого, 2026 р. / Міжнародний центр наукових досліджень. — Вінниця: ТОВ «УКРЛОГОС Груп, 2026. — 320 с.

ISBN 978-617-8582-19-7
DOI 10.62731/mcnd-27.02.2026

Викладено матеріали учасників VI Міжнародної наукової конференції «Інтелектуальний ресурс сьогодення: наукові задачі, розвиток та запитання», яка відбулася 27 лютого 2026 року у місті Черкаси.

УДК 082:001

ISBN 978-617-8582-19-7

Рисунок А.3 – Форзац з конференції №1

Рисунок А.4 – Зміст конференції №1

ЦИФРОВІ ДАНІ ЯК ІНТЕЛЕКТУАЛЬНИЙ РЕСУРС БІЗНЕСУ: ПРАВОВЕ ТА ЕТИЧНЕ РЕГУЛЮВАННЯ ВЕБСКРАПІНГУ

Караванський Владислав Володимирович
здобувач вищої освіти факультету комп'ютерно-інформаційних систем і програмної інженерії
Тернопільський національний технічний університет імені Івана Пулюя, Україна

Науковий керівник: Боднарчук Ігор Орестович
ORCID ID: 0000-0003-1443-8102
канд. техн. наук, доцент
Тернопільський національний технічний університет імені Івана Пулюя, Україна

У сучасну епоху цифрової трансформації інформація стала одним із найцінніших ресурсів економіки та ключовим чинником конкурентоспроможності бізнесу.

Маркетингові дослідження, що базуються на даних із відкритих джерел, дозволяють компаніям глибше розуміти поведінку споживачів, відстежувати тенденції ринку й оперативно приймати управлінські рішення.

Одним із найефективніших інструментів збору таких даних є вебскрапінг – технологія автоматизованого вилучення інформації з вебсайтів.

Вебскрапінг відіграє важливу роль у розвитку інформаційного бізнесу, оскільки забезпечує можливість формування великих масивів даних (Big Data), необхідних для аналітики, прогнозування попиту та побудови маркетингових стратегій. Водночас активне використання цього інструменту породжує низку правових та етичних проблем, пов'язаних із дотриманням законодавства про захист персональних даних, авторських прав, умов використання вебресурсів і принципів доброчесності у зборі інформації.

Межа між законним збором відкритих даних і порушенням прав власників контенту є часто розмитою. У зв'язку з цим дослідження етичних і правових аспектів вебскрапінгу є необхідним для формування відповідальної інформаційної культури в бізнесі, забезпечення прозорості маркетингових процесів і запобігання зловживанням у сфері цифрової аналітики.

Рисунок А.5 – Вміст тези конференції №1

Немає конкретних законів, які забороняють вебскрапінг, багато компаній використовують його легально для збору цінних даних за допомогою різних інструментів вебскрапінгу [1].

Проте завжди є винятки, на які варто звернути увагу і пам'ятати про них. Наприклад, коли ви входите в систему, інколи ви повинні погодитися з Умовами надання послуг (ToS) вебсайту, які часто забороняють автоматизований збір даних [1].

Загальнодоступні дані не завжди можна використовувати без обмежень, навіть із публічними даними ви повинні бути обережними, щоб не порушувати закони, особливо щодо авторського права [1].

Завантаження матеріалів, захищених авторським правом, таких як статті, відео чи дизайни, зазвичай є незаконним – цей матеріал захищений законами про авторське право [1].

Деякі Умови надання послуг можуть забороняти будь-який автоматичний збір даних, незалежно від цільового використання даних [1]. У цих випадках сама діяльність зі скрапінгу може бути незаконною, а не лише використання даних [1].

Щоб зрозуміти, чи є вебскрапінг законним, варто також розглядати реальні приклади з життя. Такі випадки можуть показати поточний стан галузі та куди вона може рухатися. Таке дослідження може суттєво допомогти в подальшій ідентифікації моментів, коли варто відмовитися від ідеї вебскрапінгу.

Справа Ryanair проти PR Aviation відбувалася в 2018 році [1]. Ryanair подала до суду на PR Aviation за скрапінг цін на авіаквитки, стверджуючи про порушення їхніх Умов надання послуг (ToS) [1]. Суд розглянув, чи становили Умови надання послуг Ryanair, які були представлені як угода browsewrap (посилання на умови внизу сторінки), обов'язковий договір [1]. Нідерландський суд постановив, що оскільки PR Aviation прямо не погодилася з цими умовами, дійсний договір не був укладений, тому Ryanair виграла цю справу [1].

Варто пам'ятати – якщо ваш автоматизований збір даних повинен погодитися з Умовами надання послуг конкретного ресурсу, перш ніж зібрати дані, потрібно ретельно дослідити їх і впевнитися, що така діяльність не порушує ці Умови. Завжди потрібно ставитися то процесу вебскрапінгу з доброчесністю та не виходити за межі дозволеного.

Деякі ресурси можуть навпаки дозволяти збір даних. Інколи вебсайти містять файл robots.txt, в якому вказано для автоматизованих

Рисунок А.6 – Вміст тези конференції №1

систем: з якою швидкістю, в яких розділах можна/не можна збирати дані. Або ж це також може бути дозволено в Умовах надання послуг, адже не все, що там вказано – є забороною.

Дотримуючись передового досвіду, залишаючись в курсі правових оновлень та звертаючись за юридичною консультацією, коли це необхідно, компанії можуть етично та ефективно використовувати вебскрапінг, щоб отримати цінну інформацію, мінімізуючи юридичні ризики [1].

Висновки. Вебскрапінг виступає важливою складовою інформаційного бізнесу, адже забезпечує компаніям доступ до великих масивів даних, що дозволяють оперативно реагувати на зміни попиту, відстежувати дії конкурентів і прогнозувати тренди. Проведений аналіз правових аспектів показав, що законність використання вебскрапінгу залежить від дотримання норм міжнародного та національного законодавства у сфері захисту даних, авторського права й умов користування вебресурсами.

Список використаних джерел:

1. Kaur H. What is Web Scraping and How to Use It?. GeeksforGeeks. 12.11.2025. URL: https://www.geeksforgeeks.org/blogs/what-is-web-scraping-and-how-to-use-it/ (дата звернення: 22.02.2026).

Рисунок А.7 – Вміст тези конференції №1

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

Рисунок А.8 – Титульна сторінка конференції №2

ББК 72+34 (Укр)
М34

Матеріали IX Міжнародної студентської науково - технічної конференції / Тернопіль: Тернопільський національний технічний університет ім. І.Пулюя (м. Тернопіль, 24-25 квітня 2026 р.), 2026.- 391 с.

В збірнику друкуються матеріали IX Міжнародної студентської науково-технічної конференції. Тернопіль. – ТНТУ ім. І. Пулюя (24-25 квітня 2026 р.) за наступними науковими спеціальностями:
культура і мистецтво; гуманітарні науки; соціальні та поведінкові науки; управління та адміністрування; природничі науки; математика та статистика; інформаційні технології; механічна інженерія; електрична інженерія; автоматизація та приладобудування; хімічна та біоінженерія; електроніка та телекомунікації; виробництво та технології; архітектура та будівництво; аграрні науки та продовольство; сфера обслуговування; транспорт.

Адреса конференції:
46001, м. Тернопіль, вул. Руська, 56
Тернопільський національний технічний університет ім. Івана Пулюя
e-mail: snt@tntu.edu.ua
Тернопільський національний технічний університет ім. Івана Пулюя

Рисунок А.9 – Форзац конференції №2

Рисунок А.10 – Зміст конференції №2

УДК 621.326
Караванський В.– ст. гр. СНнм-61
Тернопільський національний технічний університет імені Івана Пулюя

АНАЛІЗ МЕТОДІВ ВИЗНАЧЕННЯ ПОПУЛЯРНИХ ТОВАРНИХ ПІДКАТЕГОРІЙ ТА ПРОГНОЗУВАННЯ ЇХ ПОПУЛЯРНОСТІ

Науковий керівник: к.т.н., доцент Боднарчук І. О.

Karavanskyy V.
Ternopil Ivan Puluj National Technical University

ANALYSIS OF METHODS FOR DETERMINING POPULAR SUBCATEGORIES AND FORECASTING THEIR POPULARITY

Supervisor: PhD, Associate Professor Bodnarchuk I. O.

Ключові слова: аналіз, прогнозування, товари.
Keywords: analysis, forecasting, products.

Щоб зрозуміти, як дані змінюються з часом, використовується аналіз та прогнозування часових рядів, які допомагають відстежувати минулі закономірності та прогнозувати майбутні значення. Цей аналіз: широко використовується у фінансах, погоді, продажах та даних датчиків; зосереджується на даних, зібраних через регулярні проміжки часу; допомагає визначити тенденції, сезонність та раптові зміни; корисний для планування, прогнозування та прийняття рішень; поширені методи включають ARIMA, експоненціальне згладжування та моделі машинного навчання [1].

Часові ряди (time series) — це послідовність спостережень певного показника, які впорядковані в часі. Найчастіше їх візуалізують за допомогою лінійного графіка, де по осі X відкладається час (дні, місяці, роки тощо), а по осі Y — значення досліджуваного показника. Така візуалізація дозволяє легко побачити тренди, коливання та приховані закономірності у даних. Завдяки цьому дослідники та аналітики можуть зрозуміти, як змінюється показник з часом і які фактори можуть впливати на ці зміни [1].

Аналіз часових рядів дозволяє будувати моделі, які допомагають прогнозувати майбутні значення показників на основі історичних даних [1]. Це особливо важливо для бізнесу та економіки, де прогнозування використовується для оцінки майбутнього попиту на товари, прогнозування доходів компанії, змін цін на ринку або динаміки фінансових активів. Завдяки таким прогнозам організації можуть заздалегідь планувати свої ресурси та приймати більш обґрунтовані рішення.

Часові ряди допомагають знаходити циклічні та сезонні закономірності [1]. Наприклад, попит на певні товари може зростати у святкові періоди або змінюватися залежно від пори року. Крім того, аналіз дозволяє виявляти аномалії — незвичайні або неочікувані зміни у даних, які можуть свідчити про помилки вимірювання, технічні збої або важливі події на ринку.

Завдяки аналізу часових рядів можна виявляти ранні сигнали потенційних проблем або небезпечних тенденцій [1]. Наприклад, різке падіння продажів або незвичайні зміни в поведінці користувачів можуть бути сигналом необхідності швидкого реагування. Раннє виявлення таких змін дозволяє організаціям запобігати фінансовим втратам або іншим негативним наслідкам. Результати аналізу часових рядів

Рисунок А.11 – Вміст тези конференції №2

активно використовуються у стратегічному плануванні [1]. На їх основі компанії можуть планувати бюджети, оптимізувати кількість персоналу, керувати запасами товарів та визначати довгострокову політику розвитку. Наприклад, прогноз попиту допомагає уникнути як дефіциту товарів, так і надлишкових запасів.

Організації, які ефективно використовують аналіз часових рядів, можуть швидше реагувати на зміни ринку та адаптувати свої стратегії [1]. Це дає змогу оптимізувати операційні процеси, краще задовольняти потреби клієнтів і випереджати конкурентів у прийнятті рішень.

Часові ряди можна групувати на основі структури, часового інтервалу та статистичної поведінки. Ці категорії допомагають у виборі правильного методу прогнозування.

1. Одновимірний проти багатовимірного. Одновимірний часовий ряд реєструє лише одну змінну з плином часу, що спрощує його аналіз та моделювання. Багатовимірний часовий ряд відстежує кілька пов'язаних змінних разом, щоб показати, як вони впливають одна на одну з плином часу [1].
2. Безперервний проти дискретного часового ряду. Безперервний часовий ряд спостерігається в кожен момент або з дуже високою частотою, як-от сигнали ЕКГ або датчиків. Дискретний часовий ряд реєструється з фіксованими інтервалами, такими як щогодини, щодня або щомісяця, і є найпоширенішим форматом [1].
3. Стаціонарний проти нестаціонарного. Стаціонарний ряд має постійне середнє значення, дисперсію та закономірність з плином часу без тренду чи сезонності. Нестаціонарний ряд показує змінні закономірності, такі як тренди чи сезонність, і часто потребує трансформації перед моделюванням [1].

У контексті аналізу популярності товарних підкатегорій, дані про популярність товарів збираються через певні регулярні проміжки часу (наприклад, щодня або щомісяця), тому такі дані належать до дискретних часових рядів.

Крім того, для оцінювання популярності окремої товарної підкатегорії у часі зазвичай аналізується один основний показник (наприклад, кількість переглядів, продажів або сформований індекс популярності). У такому випадку часовий ряд можна розглядати як одновимірний, оскільки він відображає зміну одного показника з плином часу.

Водночас популярність товарів у сфері електронної комерції часто змінюється під впливом сезонних факторів, модних тенденцій та поведінки споживачів, що призводить до появи трендів та сезонних коливань у даних. Тому такі часові ряди зазвичай є нестаціонарними, оскільки їх статистичні характеристики можуть змінюватися з часом.

Таким чином, для дослідження, присвяченого аналізу методів визначення популярних товарних підкатегорій та прогнозування їх популярності, доцільно використовувати одновимірні дискретні нестаціонарні часові ряди, які найкраще відображають динаміку зміни популярності товарів у часі та дозволяють застосовувати відповідні методи прогнозування.

1. Pawan G. Time Series Analysis and Forecasting. GeeksforGeeks. Machine learning. 19.12.2025. URL: https://www.geeksforgeeks.org/machine-learning/time-series-analysis-and-forecasting/ (дата звернення: 11.03.2026).

Рисунок А.12 – Вміст тези конференції №2

Вихідний код до практичного завдання

Мова програмування: Python.

Середовище: Visual Studio Code.

Файл main.py

```
import os
import asyncio
import logging

from config import DBConfig
from scraper import Scraper
from database import Database
from dotenv import load_dotenv

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(name)s - %(message)s",
)

load_dotenv()
BASE_URL = os.getenv("BASE_URL")

async def main() -> None:
    db_cfg = DBConfig.from_env()

    async with Database(dsn=db_cfg.dsn, min_size=db_cfg.min_pool,
max_size=db_cfg.max_pool) as db:
        await db.initialise()

        scraper = Scraper.from_env(base_url=BASE_URL, db=db)
        async with scraper:
            results = await scraper.run_from_listing(
                scraper._cfg.listing_url
            )
            _print_results(results)

def _print_results(results) -> None:
    for r in results:
        status = "✓" if r.ok else "✗"
        print(f"[{status}] {r.url}")
        if r.error:
            print(f"     error   : {r.error}")
        if r.elements:
            print(f"     elements: {len(r.elements)}")
        for e in [lg for lg in r.logs if lg["log_type"] == "error"]:
            print(f"     log     : {e['description']} -
{e['additional_data']}")
```

```
if __name__ == "__main__":
    asyncio.run(main())
```

Файл config.py

```
from __future__ import annotations

import os
from dataclasses import dataclass

from dotenv import load_dotenv

load_dotenv()


def _require(key: str) -> str:
    value = os.getenv(key)
    if not value:
        raise EnvironmentError(f"Required environment variable
'{key}' is not set.")
    return value


@dataclass(frozen=True)
class DBConfig:
    host: str
    port: int
    name: str
    user: str
    password: str
    min_pool: int
    max_pool: int

    @classmethod
    def from_env(cls) -> "DBConfig":
        return cls(
            host=_require("DB_HOST"),
            port=int(os.getenv("DB_PORT", "5432")),
            name=_require("DB_NAME"),
            user=_require("DB_USER"),
            password=_require("DB_PASSWORD"),
            min_pool=int(os.getenv("DB_MIN_POOL", "2")),
            max_pool=int(os.getenv("DB_MAX_POOL", "10")),
        )

    @property
    def dsn(self) -> str:
        return
f"postgresql://{self.user}:{self.password}@{self.host}:{self.port}/{s
elf.name}"


@dataclass(frozen=True)
class ScraperEnvConfig:
    concurrency: int
    timeout: int
    max_retries: int
```

```
    retry_backoff: float
    products_table: str
    logs_table: str
    user_agent: str
    cookie: str
    listing_url:str

    @classmethod
    def from_env(cls) -> "ScraperEnvConfig":
        return cls(
            concurrency=int(os.getenv("SCRAPER_CONCURRENCY", "10")),
            timeout=int(os.getenv("SCRAPER_TIMEOUT", "40")),
            max_retries=int(os.getenv("SCRAPER_MAX_RETRIES", "3")),
            retry_backoff=float(os.getenv("SCRAPER_RETRY_BACKOFF",
"2.0")),
            products_table=os.getenv("PRODUCTS_TABLE", "products"),
            logs_table=os.getenv("LOGS_TABLE", "scrape_logs"),
            user_agent=os.getenv("USER_AGENT", ""),
            cookie=os.getenv("COOKIE", ""),
            listing_url=os.getenv("LISTING_URL", ""),
        )
```

Файл database.py

```
from __future__ import annotations

import asyncpg
import logging

from typing import Any
from pathlib import Path

logger = logging.getLogger(__name__)


class Database:
    """
    Async PostgreSQL client built on asyncpg.

    Usage
    -----
    db = Database(dsn="postgresql://user:pass@host:5432/dbname")
    await db.connect()

    await db.write("products", rows)
    rows = await db.get("products", filters={"gender": "female"})

    await db.close()

    # - or use as an async context manager -
    async with Database(dsn=...) as db:
        ...
    """

    def __init__(self, dsn: str, min_size: int = 2, max_size: int =
10) -> None:
        self._dsn = dsn
```

```
        self._min_size = min_size
        self._max_size = max_size
        self._pool: asyncpg.Pool | None = None


    # lifecycle
    async def connect(self) -> None:
        self._pool = await asyncpg.create_pool(
            dsn=self._dsn,
            min_size=self._min_size,
            max_size=self._max_size,
        )
        logger.info("Database pool created (min=%d, max=%d)",
self._min_size, self._max_size)

    async def initialise(self, sql_path: str | Path | None = None) ->
None:
        """
        Run the setup SQL script to create tables and indexes if they
don't
        exist yet.  Safe to call on every startup – all statements
are
        idempotent (IF NOT EXISTS guards).

        Parameters
        ----------
        sql_path : path to the .sql file.
                   Defaults to setup.sql sitting next to this module.
        """
        self._ensure_connected()

        if sql_path is None:
            sql_path = Path(__file__).parent / "setup.sql"

        sql = Path(sql_path).read_text(encoding="utf-8")
        async with self._pool.acquire() as conn:
            await conn.execute(sql)

        logger.info("Database initialised from %s", sql_path)

    async def close(self) -> None:
        if self._pool:
            await self._pool.close()
            logger.info("Database pool closed")

    async def __aenter__(self) -> "Database":
        await self.connect()
        return self

    async def __aexit__(self, *_) -> None:
        await self.close()


    # public interface
    async def write(
        self,
        table: str,
        rows: list[dict[str, Any]],
```

```
        *,
        on_conflict: str = "DO NOTHING",
    ) -> int:
        """
        Upsert *rows* into *table*.

        Parameters
        ----------
        table       : target table name
        rows        : list of dicts – keys must match column names
        on_conflict : raw SQL conflict clause, e.g.
                      "DO NOTHING"
                      "DO UPDATE SET price = EXCLUDED.price,
timestamp = EXCLUDED.timestamp"

        Returns the number of rows affected.
        """
        if not rows:
            return 0

        self._ensure_connected()

        columns = list(rows[0].keys())
        placeholders = ", ".join(f"${i + 1}" for i in
range(len(columns)))
        col_names = ", ".join(columns)
        sql = (
            f"INSERT INTO {table} ({col_names}) "
            f"VALUES ({placeholders}) "
            f"ON CONFLICT {on_conflict}"
        )

        records = [tuple(row[col] for col in columns) for row in
rows]

        async with self._pool.acquire() as conn:
            result = await conn.executemany(sql, records)

        # executemany returns "INSERT 0 N" – extract N
        affected = int(result.split()[-1]) if result else 0
        logger.debug("write → %s: %d row(s) affected", table,
affected)
        return affected

    async def get(
        self,
        table: str,
        *,
        columns: list[str] | None = None,
        filters: dict[str, Any] | None = None,
        order_by: str | None = None,
        limit: int | None = None,
    ) -> list[dict[str, Any]]:
        """
        Fetch rows from *table*.

        Parameters
        ----------
```

```
        columns  : columns to select (default: all)
        filters  : simple equality conditions, e.g. {"gender":
"female", "availability": "in_stock"}
        order_by : raw ORDER BY clause, e.g. "timestamp DESC"
        limit    : max rows to return

        Returns a list of dicts.
        """
        self._ensure_connected()

        select = ", ".join(columns) if columns else "*"
        sql = f"SELECT {select} FROM {table}"
        args: list[Any] = []

        if filters:
            conditions = " AND ".join(
                f"{col} = ${i + 1}" for i, col in enumerate(filters)
            )
            sql += f" WHERE {conditions}"
            args = list(filters.values())

        if order_by:
            sql += f" ORDER BY {order_by}"
        if limit:
            sql += f" LIMIT {limit}"

        async with self._pool.acquire() as conn:
            rows = await conn.fetch(sql, *args)

        result = [dict(row) for row in rows]
        logger.debug("get ← %s: %d row(s) returned", table,
len(result))
        return result

    async def execute(self, sql: str, *args: Any) -> str:
        """Run arbitrary SQL (CREATE TABLE, DELETE, etc.)."""
        self._ensure_connected()
        async with self._pool.acquire() as conn:
            return await conn.execute(sql, *args)

    async def execute_fetch(self, sql: str, *args: Any) ->
list[dict[str, Any]]:
        """Run arbitrary SQL that returns rows (SELECT, WITH …).
Returns list of dicts."""
        self._ensure_connected()
        async with self._pool.acquire() as conn:
            rows = await conn.fetch(sql, *args)
        return [dict(row) for row in rows]

    # internals
    def _ensure_connected(self) -> None:
        if self._pool is None:
            raise RuntimeError("Database.connect() has not been
called yet.")
```

Файл parser.py

```
from __future__ import annotations

import re
import json
import pandas as pd

from bs4 import BeautifulSoup
from datetime import datetime
from dataclasses import dataclass, field
from urllib.parse import urljoin, urlparse


# helpers
def _now_ts() -> datetime:
    return datetime.now()


def _make_error(description: str, url: str, additional_data=None) ->
dict:
    return {
        "log_type": "error",
        "description": description,
        "link": url,
        "timestamp": _now_ts(),
        "additional_data": additional_data,
    }


def _parse_condition(raw: str) -> str | None:
    value = raw.replace("https://schema.org/", "")
    if value in ("DamagedCondition", "UsedCondition"):
        return "used"
    if value == "RefurbishedCondition":
        return "refurbished"
    if value == "NewCondition":
        return "new"
    return None


def _parse_availability(raw: str) -> str | None:
    value = raw.replace("https://schema.org/", "")
    if value in ("BackOrder", "Discontinued", "OutOfStock",
"PreOrder", "PreSale", "SoldOut", "InStoreOnly"):
        return "out_of_stock"
    if value in ("InStock", "LimitedAvailability", "OnlineOnly"):
        return "in_stock"
    return None


def _parse_price_string(value: float | None, currency: str) -> str |
None:
    if isinstance(value, (int, float)) and value > 0:
        return f"{value} {currency}"
    return None
```

```
def _resolve_sale_price(
    was_price_str: str | None,
    current_price: float | None,
) -> tuple[float | None, float | None]:
    """Return (final_price, sale_price). Promotes current_price to
sale if was > current."""
    if not was_price_str or current_price is None:
        return current_price, None
    was = float(was_price_str.replace("R", ""))
    if was > current_price:
        return was, current_price
    return current_price, None


def _parse_images(data: dict) -> tuple[str | None, list[str] | None]:
    images = data.get("image")
    if not images:
        return None, None
    big = [i.replace("ProductImages/", "ProductImages/Big/") for i in
images]
    return big[0], big


# Parser class
class ProductParser:
    """
    Stateless HTML parser for a single product page.

    Usage
    -----
    parser = ProductParser()
    logs, elements = parser.parse(html_code, url)
    """

    BAD_CATEGORIES = ("gift", "gifts", "gifting")


    # public interface
    def parse(self, html_code: str, url: str) -> tuple[list[dict],
list[dict]]:
        """
        Parse raw HTML and return (logs, elements).
        Returns an empty elements list on any fatal error.
        """
        self._logs: list[dict] = []
        self._url = url

        soup = BeautifulSoup(html_code, "html.parser")

        product_data = self._extract_product_data(soup)
        if product_data is None:
            return self._logs, []

        breadcrumbs_values = self._extract_breadcrumbs(soup,
product_data)
        if breadcrumbs_values is None:
            return self._logs, []
```

```
        if self._has_bad_category(breadcrumbs_values):
            return self._logs, []

        if self._is_kids_product(product_data, breadcrumbs_values):
            return self._logs, []

        shared = self._build_shared_fields(soup, product_data,
breadcrumbs_values)
        if shared is None:
            return self._logs, []

        elements = self._build_elements(product_data, shared)
        self._logs.append({
            "log_type": "success",
            "description": "Objects successfully created",
            "link": url,
            "timestamp": _now_ts(),
            "additional_data": f"{len(elements)} element(s)",
        })
        return self._logs, elements


    # private helpers
    def _error(self, description: str, additional_data=None) -> dict:
        entry = _make_error(description, self._url, additional_data)
        self._logs.append(entry)
        return entry

    # script extraction
    def _extract_product_data(self, soup: BeautifulSoup) -> dict |
None:
        all_scripts = soup.find_all("script",
type="application/ld+json")
        if not all_scripts:
            self._error("No scripts data")
            return None

        product_script = breadcrumbs_script = None
        for script in all_scripts:
            if not (script and script.contents):
                continue
            text = script.contents[0].replace(" ", "")
            if '"@type":"ProductGroup"' in text:
                product_script = script.contents[0]
            if '"@type":"BreadcrumbList"' in text:
                breadcrumbs_script = script.contents[0]

        if not product_script:
            self._error("No product script")
            return None

        data = json.loads(product_script)
        if isinstance(data, list):
            if len(data) > 1:
                self._error("Unexpected number of product data")
            data = data[0]

        # attach breadcrumbs_script for later use
```

```
        data["_breadcrumbs_script"] = breadcrumbs_script
        return data

    # breadcrumbs

    def _extract_breadcrumbs(self, soup: BeautifulSoup, product_data:
dict) -> list[str] | None:
        breadcrumbs_script = product_data.pop("_breadcrumbs_script",
None)
        breadcrumbs_data = json.loads(breadcrumbs_script) if
breadcrumbs_script else None
        values = (
            [item["item"]["name"] for item in
breadcrumbs_data["itemListElement"]]
            if breadcrumbs_data else None
        )
        if not values:
            values = [product_data["Category"]] if
product_data.get("Category") else None
        if not values:
            self._error("No breadcrumbs values")
            return None
        return values

    def _has_bad_category(self, breadcrumbs: list[str]) -> bool:
        lower = [v.lower() for v in breadcrumbs]
        for word in self.BAD_CATEGORIES:
            if word in lower:
                self._error("Redundant element", f"Found '{word}' in
breadcrumbs: {breadcrumbs}")
                return True
        return False

    def _is_kids_product(self, product_data: dict, breadcrumbs:
list[str]) -> bool:
        # Hook: override or inject an external check_for_kids
function if needed.
        return False

    # shared fields

    def _build_shared_fields(
        self,
        soup: BeautifulSoup,
        product_data: dict,
        breadcrumbs_values: list[str],
    ) -> dict | None:
        lower = [v.lower() for v in breadcrumbs_values]
        gender = "male" if any(v in lower for v in ("men",
"menswear")) else "female"

        offers = product_data.get("offers", {})
        price_currency = offers.get("priceCurrency") or None
        if not price_currency:
            self._error("No currency data")
            return None

        was_price_str = self._scrape_was_price(soup)
```

```
        raw_price = float(offers["price"].replace(',', '')) if
offers.get("price") else None
        main_price, main_sale_price =
_resolve_sale_price(was_price_str, raw_price)

        main_image_link, main_additional_image_link =
_parse_images(product_data)

        return {
            "item_group_id": product_data["productID"],
            "title": product_data["name"],
            "description": (product_data.get("description") or
"").strip() or None,
            "brand": product_data["brand"],
            "gender": gender,
            "age_group": "adult",
            "product_type": ">".join(breadcrumbs_values),
            "main_color": product_data.get("color") or None,
            "main_size": product_data.get("size") or None,
            "main_image_link": main_image_link,
            "main_additional_image_link": main_additional_image_link,
            "was_price_str": was_price_str,
            "main_price": main_price,
            "main_sale_price": main_sale_price,
            "main_condition":
_parse_condition(offers["itemCondition"]) if
offers.get("itemCondition") else None,
            "main_availability":
_parse_availability(offers["availability"]) if
offers.get("availability") else None,
            "price_currency": price_currency,
            "main_offers": offers,
        }

    @staticmethod
    def _scrape_was_price(soup: BeautifulSoup) -> str | None:
        bar = soup.find("div", class_="product-info__pricing-bar")
        if not bar:
            return None
        was_block = bar.find("div", class_="pricing-block--was")
        if not was_block:
            return None
        return (
            was_block.get_text()
            .replace("$", "").replace("€", "").replace("£", "")
            .replace(",", ".").strip()
        )

    # element builder

    def _build_elements(self, product_data: dict, shared: dict) ->
list[dict]:
        variants = product_data.get("hasVariant")
        if not variants:
            return [self._build_element(None, product_data, shared)]
        return [self._build_element(v, product_data, shared) for v in
variants]
```

```
    def _build_element(
        self,
        variant: dict | None,
        product_data: dict,
        shared: dict,
    ) -> dict:
        s = shared

        if variant:
            sku = variant["sku"]
            color = variant.get("color") or s["main_color"]
            size = variant.get("size") or s["main_size"]
            image_link, additional_image_link =
_parse_images(variant)
            if not image_link:
                image_link, additional_image_link =
s["main_image_link"], s["main_additional_image_link"]

            v_offers = variant.get("offers", {})
            raw_price = float(v_offers["price"].replace(',', '')) if
v_offers.get("price") else s["main_price"]
            price_val, sale_price_val =
_resolve_sale_price(s["was_price_str"], raw_price)
            condition = _parse_condition(v_offers["itemCondition"])
if v_offers.get("itemCondition") else s["main_condition"]
            availability =
_parse_availability(v_offers["availability"]) if
v_offers.get("availability") else None
            gtin = v_offers.get("gtin13") or None
        else:
            sku = product_data["sku"]
            color = s["main_color"]
            size = s["main_size"]
            image_link, additional_image_link = s["main_image_link"],
s["main_additional_image_link"]
            price_val, sale_price_val = s["main_price"],
s["main_sale_price"]
            condition = s["main_condition"]
            availability = s["main_availability"]
            gtin = s["main_offers"].get("gtin13") or None

        # price validation logs
        if price_val is None:
            self._error("No price data")
        elif price_val == 0:
            self._error("Price is zero")
        if sale_price_val == 0:
            self._error("Sale price is zero")
        if not availability:
            self._error("No availability data")

        return {
            "id": sku,
            "item_group_id": s["item_group_id"],
            "mpn": s["item_group_id"],
            "gtin": gtin,
            "title": s["title"],
            "description": s["description"],
```

```
            "image_link": image_link,
            "additional_image_link": additional_image_link,
            "link": self._url,
            "gender": s["gender"],
            "age_group": s["age_group"],
            "brand": s["brand"],
            "color": color,
            "size": size,
            "availability": availability,
            "price": _parse_price_string(price_val,
s["price_currency"]),
            "sale_price": _parse_price_string(sale_price_val,
s["price_currency"]),
            "condition": condition,
            "product_type": s["product_type"],
            "timestamp": _now_ts(),
        }


@dataclass
class ListingPage:
    """Parsed result from a single listing/category page."""
    source_url: str
    product_urls: list[str] = field(default_factory=list)
    next_page_url: str | None = None
    total_products: int | None = None   # if the site advertises a
count
    current_page: int | None = None
    error: str | None = None

    @property
    def ok(self) -> bool:
        return self.error is None and bool(self.product_urls)


class ListingParser:
    """
    Parses a category / search-results page and extracts:
      - individual product URLs
      - next-page URL (for pagination)
      - advertised total product count (if present)

    The class ships with sensible defaults that match Evans / Yours
Clothing
    style sites. Override the selector constants on a subclass to
adapt to
    other shops without touching any logic.

    Usage
    -----
    parser = ListingParser(base_url="https://www.evans.co.uk")
    listing = parser.parse(html,
url="https://www.evans.co.uk/dresses")

    # listing.product_urls  → ["/dress-a-p", "/dress-b-p", ...]
    # listing.next_page_url →
"https://www.evans.co.uk/dresses?page=2"
    """
```

```
    # Selector constants – override in a subclass for a different
site
    # CSS selector that matches each product card / tile on the
listing page
    PRODUCT_LINK_SELECTOR = "a.associated-product__meta"

    # CSS selector for the "next page" link
    NEXT_PAGE_SELECTOR: str = "a[rel='next'], a.pagination__next,
li.next > a"

    # CSS selector (or None to skip) for the total-count element
    TOTAL_COUNT_SELECTOR: str | None = "span.product-count,
span.results-count"

    # Regex applied to the count element's text to extract the
integer
    TOTAL_COUNT_PATTERN: re.Pattern = re.compile(r"(\d[\d,]*)")

    # URL path suffix that every product URL must contain (None = no
filter)
    PRODUCT_URL_SUFFIX: str | None = "-p"


    # public interface
    def __init__(self, base_url: str) -> None:
        """
        Parameters
        ----------
        base_url : scheme + host, e.g. "https://www.evans.co.uk"
                   Used to turn relative hrefs into absolute URLs.
        """
        parsed = urlparse(base_url)
        self._origin = f"{parsed.scheme}://{parsed.netloc}"

    def parse(self, html: str, url: str) -> ListingPage:
        """
        Parse raw listing-page HTML.

        Returns a ListingPage dataclass with product_urls and
next_page_url.
        """
        result = ListingPage(source_url=url)
        try:
            soup = BeautifulSoup(html, "html.parser")
            result.product_urls = self._extract_product_urls(soup)
            result.next_page_url = self._extract_next_page(soup)
            result.total_products = self._extract_total_count(soup)
            result.current_page = self._extract_current_page(url)

            if not result.product_urls:
                result.error = "No product URLs found"

        except Exception as exc:
            result.error = f"ListingParser error: {exc}"

        return result
```

```
    # private helpers
    def _extract_product_urls(self, soup: BeautifulSoup) ->
list[str]:
        anchors = soup.select(self.PRODUCT_LINK_SELECTOR)
        seen: set[str] = set()
        urls: list[str] = []

        for tag in anchors:
            href = tag.get("href", "")
            if not href:
                continue
            if self.PRODUCT_URL_SUFFIX and self.PRODUCT_URL_SUFFIX
not in href:
                continue
            absolute = self._to_absolute(href)
            if absolute not in seen:
                seen.add(absolute)
                urls.append(absolute)

        return urls

    def _extract_next_page(self, soup: BeautifulSoup) -> str | None:
        tag = soup.select_one(self.NEXT_PAGE_SELECTOR)
        if not tag:
            return None
        href = tag.get("href", "")
        return self._to_absolute(href) if href else None

    def _extract_total_count(self, soup: BeautifulSoup) -> int |
None:
        if not self.TOTAL_COUNT_SELECTOR:
            return None
        tag = soup.select_one(self.TOTAL_COUNT_SELECTOR)
        if not tag:
            return None
        match = self.TOTAL_COUNT_PATTERN.search(tag.get_text())
        if match:
            return int(match.group(1).replace(",", ""))
        return None

    @staticmethod
    def _extract_current_page(url: str) -> int | None:
        match = re.search(r"[?&]page=(\d+)", url)
        return int(match.group(1)) if match else 1

    def _to_absolute(self, href: str) -> str:
        return urljoin(self._origin, href)
```

Файл scraper.py

```
from __future__ import annotations

import asyncio
import logging
from dataclasses import dataclass, field
```

```
from curl_cffi.requests import AsyncSession, RequestsError

from config import ScraperEnvConfig
from database import Database
from parser import ListingPage, ListingParser, ProductParser

logger = logging.getLogger(__name__)


# config
@dataclass
class ScraperConfig:
    """All tuneable knobs for the scraper in one place."""
    listing_url: str

    # concurrency
    concurrency: int = 10           # max simultaneous requests
    request_timeout: int = 40       # seconds per request

    # retry
    max_retries: int = 3
    retry_backoff: float = 2.0      # seconds (doubles each attempt)

    # db
    db_table: str = "products"
    db_conflict_clause: str = "DO NOTHING"

    # http
    headers: dict[str, str] = field(default_factory=dict)
    impersonate: str = "chrome110"  # curl_cffi browser fingerprint


DEFAULT_HEADERS = {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br, zstd",
    "Accept-Language": "en-US,en;q=0.9",
}


# result container
@dataclass
class ScrapeResult:
    url: str
    status_code: int | None = None
    logs: list[dict] = field(default_factory=list)
    elements: list[dict] = field(default_factory=list)
    error: str | None = None   # network / unexpected error message

    @property
    def ok(self) -> bool:
        return self.error is None and bool(self.elements)


# scraper

class Scraper:
    """
```

```
    Async scraper that fetches URLs concurrently, parses each page
with
    *parser*, and writes results to *db*.

    Usage
    -----
    config = ScraperConfig(concurrency=15, headers={"Cookie": "..."})
    parser = ProductParser()
    listing_parser =
ListingParser(base_url="https://www.evans.co.uk")
    db     = Database(dsn="postgresql://...")

    scraper = Scraper(parser=parser, db=db, config=config,
                      listing_parser=listing_parser)

    async with scraper:
        # scrape a known list of product URLs
        results = await scraper.run(urls)

        # - or - crawl a category page and scrape everything on it
        results = await scraper.run_from_listing(listing_url)

        # - or - follow pagination automatically
        results = await scraper.run_from_listing(listing_url,
follow_pages=True)
    """

    def __init__(
        self,
        parser: ProductParser,
        db: Database,
        config: ScraperConfig | None = None,
        listing_parser: ListingParser | None = None,
    ) -> None:
        self._parser = parser
        self._db = db
        self._cfg = config or ScraperConfig()
        self._listing_parser = listing_parser
        self._session: AsyncSession | None = None


    # factory
    @classmethod
    def from_env(
        cls,
        base_url: str,
        db: Database,
    ) -> "Scraper":
        """
        Build a Scraper from environment variables (via .env /
ScraperEnvConfig).

        Parameters
        ----------
        base_url : used by ListingParser to resolve relative hrefs
        db       : already-constructed Database instance
        """
        env = ScraperEnvConfig.from_env()
```

```
        config = ScraperConfig(
            concurrency=env.concurrency,
            request_timeout=env.timeout,
            max_retries=env.max_retries,
            retry_backoff=env.retry_backoff,
            db_table=env.products_table,
            headers={
                "User-Agent": env.user_agent,
                "Cookie": env.cookie,
            },
            listing_url=env.listing_url,
        )
        return cls(
            parser=ProductParser(),
            db=db,
            config=config,
            listing_parser=ListingParser(base_url=base_url),
        )


    # lifecycle
    async def _open(self) -> None:
        headers = {**DEFAULT_HEADERS, **self._cfg.headers}
        self._session = AsyncSession(
            headers=headers,
            impersonate=self._cfg.impersonate,
        )
        await self._db.connect()

    async def _close(self) -> None:
        if self._session:
            await self._session.close()
        await self._db.close()

    async def __aenter__(self) -> "Scraper":
        await self._open()
        return self

    async def __aexit__(self, *_) -> None:
        await self._close()


    # public interface
    async def run(self, urls: list[str]) -> list[ScrapeResult]:
        """
        Scrape all *urls* with bounded concurrency.
        Returns one ScrapeResult per URL in the same order.
        """
        semaphore = asyncio.Semaphore(self._cfg.concurrency)
        tasks = [self._bounded_scrape(url, semaphore) for url in
urls]
        results = await asyncio.gather(*tasks,
return_exceptions=False)

        ok = sum(1 for r in results if r.ok)
        logger.info("Scrape complete - %d/%d succeeded", ok,
len(urls))
        return results
```

```
    async def run_from_listing(
        self,
        listing_url: str,
        *,
        follow_pages: bool = False,
        max_pages: int = 50,
    ) -> list[ScrapeResult]:
        """
        Fetch a category/listing page, collect all product URLs from
it,
        then scrape every product.

        Parameters
        ----------
        listing_url  : URL of the first (or only) listing page
        follow_pages : if True, follow next-page links until
exhausted
        max_pages    : safety cap on the number of pages to crawl

        Returns a flat list of ScrapeResults (one per product URL).
        """
        if self._listing_parser is None:
            raise RuntimeError("listing_parser was not provided to
the Scraper.")

        product_urls: list[str] = []
        current_url: str | None = listing_url
        pages_fetched = 0

        while current_url and pages_fetched < max_pages:
            logger.info("Fetching listing page %d: %s", pages_fetched
+ 1, current_url)
            html, status = await self._fetch_with_retries(
                current_url, ScrapeResult(url=current_url)
            )
            if html is None:
                logger.error("Failed to fetch listing page: %s",
current_url)
                break

            listing: ListingPage = self._listing_parser.parse(html,
current_url)
            if not listing.ok:
                logger.warning("Listing parse issue (%s): %s",
current_url, listing.error)
                break

            product_urls.extend(listing.product_urls)
            pages_fetched += 1
            logger.info(
                "Page %d - found %d product URLs (total so far: %d)",
                pages_fetched,
                len(listing.product_urls),
                len(product_urls),
            )
```

```
            current_url = listing.next_page_url if follow_pages else
None

        logger.info(
            "Listing crawl done – %d page(s), %d product URL(s)",
            pages_fetched, len(product_urls),
        )
        return await self.run(product_urls)


    # private pipeline
    async def _bounded_scrape(
        self,
        url: str,
        semaphore: asyncio.Semaphore,
    ) -> ScrapeResult:
        async with semaphore:
            return await self._scrape_one(url)

    async def _scrape_one(self, url: str) -> ScrapeResult:
        result = ScrapeResult(url=url)

        # 1. fetch with retries
        html, status = await self._fetch_with_retries(url, result)
        if html is None:
            return result
        result.status_code = status

        # 2. parse
        try:
            logs, elements = self._parser.parse(html, url)
        except Exception as exc:
            result.error = f"Parser error: {exc}"
            logger.exception("Parser crashed on %s", url)
            return result

        result.logs = logs
        result.elements = elements

        if not elements:
            logger.warning("No elements parsed for %s", url)
            return result

        # 3. persist
        try:
            await self._db.write(
                self._cfg.db_table,
                elements,
                on_conflict=self._cfg.db_conflict_clause,
            )
        except Exception as exc:
            result.error = f"DB write error: {exc}"
            logger.exception("DB write failed for %s", url)

        return result

    async def _fetch_with_retries(
        self,
```

```
        url: str,
        result: ScrapeResult,
    ) -> tuple[str | None, int | None]:
        """
        Attempt the HTTP GET up to max_retries times.
        Returns (html_text, status_code) or (None, None) on failure.
        """
        backoff = self._cfg.retry_backoff

        for attempt in range(1, self._cfg.max_retries + 1):
            try:
                resp = await self._session.get(
                    url,
                    timeout=self._cfg.request_timeout,
                )
                if resp.status_code == 200:
                    return resp.text, resp.status_code
                logger.warning(
                    "Attempt %d/%d – HTTP %d for %s",
                    attempt, self._cfg.max_retries, resp.status_code,
url,
                )

            except (RequestsError, asyncio.TimeoutError) as exc:
                logger.warning(
                    "Attempt %d/%d – network error for %s: %s",
                    attempt, self._cfg.max_retries, url, exc,
                )

            if attempt < self._cfg.max_retries:
                await asyncio.sleep(backoff)
                backoff *= 2

        result.error = f"All {self._cfg.max_retries} attempts failed"
        return None, None
```

Файл analysis.py

```
from __future__ import annotations

import logging
import warnings
from dataclasses import dataclass, field
from itertools import product as iter_product
from typing import Literal

import numpy as np
import pandas as pd
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller, kpss

from database import Database

logger = logging.getLogger(__name__)
warnings.filterwarnings("ignore")   # suppress statsmodels
convergence noise
```

```
# ── result dataclasses ──────────────────────────────────────────


@dataclass
class StationarityResult:
    adf_statistic: float
    adf_pvalue: float
    adf_n_lags: int
    adf_is_stationary: bool             # p < 0.05  →  reject unit
root

    kpss_statistic: float
    kpss_pvalue: float
    kpss_n_lags: int
    kpss_is_stationary: bool            # p > 0.05  →  fail to reject
stationarity

    is_stationary: bool                 # both tests agree
    transformations: list[str]          # applied before testing
    recommendation: str


@dataclass
class DecompositionResult:
    trend: pd.Series
    seasonal: pd.Series
    residual: pd.Series
    observed: pd.Series
    decomp_period: int
    method: str


@dataclass
class ForecastResult:
    forecast: pd.Series
    conf_int_lower: pd.Series
    conf_int_upper: pd.Series
    model_name: str
    model_order: str                    # human-readable, e.g.
"SARIMA(1,1,1)(1,1,1,7)"
    aic: float


@dataclass
class AnalysisResult:
    subcategory: str
    metric: str
    series: pd.Series                   # original series
    transformed_series: pd.Series       # series used for modelling
    stationarity: StationarityResult
    decomposition: DecompositionResult
    forecast: ForecastResult


# ── analyser ────────────────────────────────────────────────────
```

```
class TimeSeriesAnalyzer:
    """
    Full pipeline for univariate discrete non-stationary time series
analysis
    of product subcategory popularity data.

    Pipeline per subcategory
    ------------------------
    1. Load raw series from DB
    2. Test stationarity (ADF + KPSS)
    3. Apply transformations until stationary (log → differencing)
    4. STL decomposition on original series
    5. SARIMA grid-search on transformed series
    6. Persist results to DB tables + refresh materialized views

    Usage
    -----
    analyser = TimeSeriesAnalyzer(db)

    result = await analyser.run(
        subcategory="Dresses",
        metric="views",
        forecast_periods=30,
    )
    """

    # SARIMA grid search space (p, d, q) × (P, D, Q, s)
    _AR_MA_RANGE = range(0, 3)          # p, q, P, Q
    _DIFF_RANGE  = range(0, 2)          # d, D
    _MAX_MODELS  = 20                   # cap on candidates evaluated

    def __init__(self, db: Database) -> None:
        self._db = db

    # ------------------------------------------------------------------
---
    # public interface
    # ------------------------------------------------------------------
---

    async def run(
        self,
        subcategory: str,
        metric: str,
        *,
        forecast_periods: int = 30,
        seasonal_period: int | None = None,
        refresh_views: bool = True,
    ) -> AnalysisResult:
        """
        Run the full analysis pipeline for one subcategory / metric
pair.

        Parameters
        ----------
        subcategory      : e.g. "Dresses"
        metric           : e.g. "views", "sales"
```

```
        forecast_periods : how many future periods to predict
        seasonal_period  : override auto-detected period (7, 30,
365…)
        refresh_views    : call refresh_ts_views() after writing
results
        """
        logger.info("Starting analysis: %s / %s", subcategory,
metric)

        series = await self._load_series(subcategory, metric)
        if series.empty or len(series) < 14:
            raise ValueError(
                f"Not enough data for {subcategory}/{metric} "
                f"(got {len(series)} points, need ≥ 14)."
            )

        # ── step 1: stationarity + transformation
──────────────────────
        transformed, stationarity = self._make_stationary(series)

        # ── step 2: decomposition (on original series)
─────────────────
        s_period = seasonal_period or self._detect_period(series)
        decomposition = self._decompose(series, s_period)

       # ── step 3: forecast
──────────────────────────────────────────
        # Використовуємо оригінальні дані, оскільки SARIMAX сама
робить differencing (параметри d та D).
        # Залишаємо лише логарифм, якщо тести показали його
необхідність.
        if "log" in stationarity.transformations:
            model_series = np.log(series)
        else:
            model_series = series

        forecast = self._fit_and_forecast(model_series, s_period,
forecast_periods)

        # Робимо зворотне перетворення (експоненту), якщо брали
логарифм
        if "log" in stationarity.transformations:
            forecast.forecast = np.exp(forecast.forecast)
            forecast.conf_int_lower = np.exp(forecast.conf_int_lower)
            forecast.conf_int_upper = np.exp(forecast.conf_int_upper)

        result = AnalysisResult(
            subcategory=subcategory,
            metric=metric,
            series=series,
            transformed_series=transformed,
            stationarity=stationarity,
            decomposition=decomposition,
            forecast=forecast,
        )

        # ── step 4: persist
──────────────────────────────────────────
```

```
        await self._persist(result)
        if refresh_views:
            await self._db.execute("SELECT refresh_ts_views()")
            logger.info("Materialized views refreshed")

        logger.info("Analysis complete: %s / %s", subcategory,
metric)
        return result

    async def run_many(
        self,
        pairs: list[tuple[str, str]],
        **kwargs,
    ) -> list[AnalysisResult]:
        """Run analysis for multiple (subcategory, metric) pairs
sequentially.
        Always defers the materialized-view refresh to a single call
at the end."""
        # Remove refresh_views from kwargs so we can control it
ourselves below.
        kwargs.pop("refresh_views", None)

        results = []
        for subcategory, metric in pairs:
            try:
                r = await self.run(subcategory, metric,
refresh_views=False, **kwargs)
                results.append(r)
            except Exception as exc:
                logger.error("Failed %s/%s: %s", subcategory, metric,
exc)

        # Single refresh after all pairs are done.
        await self._db.execute("SELECT refresh_ts_views()")
        return results

    # ------------------------------------------------------------------
---
    # data loading
    # ------------------------------------------------------------------
---

    async def _load_series(self, subcategory: str, metric: str) ->
pd.Series:
        rows = await self._db.get(
            "popularity_metrics",
            columns=["period", "value"],
            filters={"subcategory": subcategory, "metric": metric},
            order_by="period ASC",
        )
        if not rows:
            return pd.Series(dtype=float)
        df = pd.DataFrame(rows)
        df["period"] = pd.to_datetime(df["period"])
        series = df.set_index("period")["value"].astype(float)
        series.name = f"{subcategory}_{metric}"

        series = series.iloc[:-1]
```

```
        return series

    # ------------------------------------------------------------------
---
    # stationarity
    # ------------------------------------------------------------------
---

    def _make_stationary(
        self,
        series: pd.Series,
    ) -> tuple[pd.Series, StationarityResult]:
        """
        Apply transformations (log → 1st diff → 2nd diff) until the
series
        passes both ADF and KPSS, or until we run out of options.
        Returns (transformed_series, StationarityResult).
        """
        transformations: list[str] = []
        s = series.copy().dropna()

        # try each stage; stop as soon as stationary
        for transform in ("none", "log", "diff_1", "diff_2"):
            if transform == "log":
                if (s <= 0).any():
                    continue
                s = np.log(s)
                transformations.append("log")
            elif transform == "diff_1":
                s = s.diff().dropna()
                transformations.append("diff_1")
            elif transform == "diff_2":
                s = s.diff().dropna()
                transformations.append("diff_2")

            result = self._test_stationarity(s, transformations)
            if result.is_stationary:
                break

        return s, result

    @staticmethod
    def _test_stationarity(
        series: pd.Series,
        transformations: list[str],
    ) -> StationarityResult:
        # ADF - H0: unit root exists (non-stationary)
        adf_out = adfuller(series, autolag="AIC")
        adf_stat, adf_p, adf_lags = adf_out[0], adf_out[1],
adf_out[2]
        adf_stationary = adf_p < 0.05

        # KPSS - H0: series IS stationary
        kpss_out = kpss(series, regression="c", nlags="auto")
        kpss_stat, kpss_p, kpss_lags = kpss_out[0], kpss_out[1],
kpss_out[2]
        kpss_stationary = kpss_p > 0.05
```

```
        is_stationary = adf_stationary and kpss_stationary

        if is_stationary:
            recommendation = "Series is stationary. Ready for
modelling."
        elif adf_stationary and not kpss_stationary:
            recommendation = "ADF rejects unit root but KPSS rejects
stationarity. Possible trend-stationarity - try detrending."
        elif not adf_stationary and kpss_stationary:
            recommendation = "ADF fails to reject unit root. Apply
differencing."
        else:
            recommendation = "Both tests indicate non-stationarity.
Apply log transform and differencing."

        return StationarityResult(
            adf_statistic=float(adf_stat),
            adf_pvalue=float(adf_p),
            adf_n_lags=int(adf_lags),
            adf_is_stationary=adf_stationary,
            kpss_statistic=float(kpss_stat),
            kpss_pvalue=float(kpss_p),
            kpss_n_lags=int(kpss_lags),
            kpss_is_stationary=kpss_stationary,
            is_stationary=is_stationary,
            transformations=list(transformations),
            recommendation=recommendation,
        )

    # ------------------------------------------------------------------
---
    # decomposition
    # ------------------------------------------------------------------
---

    @staticmethod
    def _detect_period(series: pd.Series) -> int:
        """Infer seasonal period from the DatetimeIndex frequency."""
        freq = pd.infer_freq(series.index)
        if freq is None:
            return 7          # default to weekly
        freq = freq.upper()
        if "D" in freq:
            return 7
        if "W" in freq:
            return 52
        if "M" in freq or "MS" in freq:
            return 12
        if "Q" in freq:
            return 4
        if "A" in freq or "Y" in freq:
            return 1
        return 7

    @staticmethod
    def _decompose(series: pd.Series, period: int) ->
DecompositionResult:
```

```
        """STL decomposition - robust to outliers and handles any
period."""
        # STL needs at least 2 full seasonal cycles
        if len(series) < 2 * period:
            period = max(2, len(series) // 2)

        stl = STL(series.dropna(), period=period, robust=True)
        fit = stl.fit()

        return DecompositionResult(
            observed=pd.Series(fit.observed,
index=series.dropna().index),
            trend=pd.Series(fit.trend, index=series.dropna().index),
            seasonal=pd.Series(fit.seasonal,
index=series.dropna().index),
            residual=pd.Series(fit.resid,
index=series.dropna().index),
            decomp_period=period,
            method="STL",
        )

    # ------------------------------------------------------------------
---
    # forecasting
    # ------------------------------------------------------------------
---

    def _fit_and_forecast(
        self,
        series: pd.Series,
        seasonal_period: int,
        n_periods: int,
    ) -> ForecastResult:
        """
        Grid-search SARIMA orders by AIC, fit the best model, return
forecast.
        Seasonal component is included only if series is long enough.
        """
        use_seasonal = len(series) >= 2 * seasonal_period
        best_model, best_aic, best_order = None, np.inf, None

        candidates = self._build_candidate_orders(seasonal_period,
use_seasonal)
        for (p, d, q), (P, D, Q, s) in candidates:
            try:
                mdl = SARIMAX(
                    series,
                    order=(p, d, q),
                    seasonal_order=(P, D, Q, s),
                    enforce_stationarity=False,
                    enforce_invertibility=False,
                ).fit(disp=False)
                if mdl.aic < best_aic:
                    best_aic = mdl.aic
                    best_model = mdl
                    best_order = ((p, d, q), (P, D, Q, s))
            except Exception:
                continue
```

```
        if best_model is None:
            raise RuntimeError("SARIMA grid search produced no valid
models.")

        pred = best_model.get_forecast(steps=n_periods)
        summary = pred.summary_frame(alpha=0.05)

        # build future date index
        last_date = series.index[-1]
        freq = pd.infer_freq(series.index) or "D"
        future_index = pd.date_range(start=last_date,
periods=n_periods + 1, freq=freq)[1:]

        (p, d, q), (P, D, Q, s) = best_order
        order_str = f"SARIMA({p},{d},{q})({P},{D},{Q},{s})"

        return ForecastResult(
            forecast=pd.Series(summary["mean"].values,
index=future_index),
            conf_int_lower=pd.Series(summary["mean_ci_lower"].values,
index=future_index),
            conf_int_upper=pd.Series(summary["mean_ci_upper"].values,
index=future_index),
            model_name="SARIMA",
            model_order=order_str,
            aic=float(best_aic),
        )

    def _inverse_transform(
        self,
        forecast: pd.Series,
        original_series: pd.Series,
        transformations: list[str],
    ) -> pd.Series:
        """Інвертує трансформації у зворотному порядку."""
        result = forecast.copy()

        steps = list(reversed(transformations))
        for step in steps:
            if step == "diff_2":
                # додати останню першу різницю оригінального ряду
                last_diff1 = original_series.diff().iloc[-1]
                result = result.cumsum() + last_diff1
            elif step == "diff_1":
                # додати останнє оригінальне значення
                last_val = original_series.iloc[-1]
                result = result.cumsum() + last_val
            elif step == "log":
                result = np.exp(result)

        return result

    def _build_candidate_orders(
        self,
        s: int,
        use_seasonal: bool,
    ) -> list[tuple]:
```

```
        """Return a capped list of (non-seasonal, seasonal) order
pairs."""
        non_seasonal = list(iter_product(self._AR_MA_RANGE,
self._DIFF_RANGE, self._AR_MA_RANGE))
        seasonal = (
            list(iter_product(self._AR_MA_RANGE, self._DIFF_RANGE,
self._AR_MA_RANGE, [s]))
            if use_seasonal else [(0, 0, 0, 0)]
        )
        pairs = list(iter_product(non_seasonal, seasonal))
        # prioritise low-order models; cap total candidates
        pairs.sort(key=lambda x: sum(x[0]) + sum(x[1][:3]))
        return pairs[: self._MAX_MODELS]

    # ------------------------------------------------------------------
---
    # persistence
    # ------------------------------------------------------------------
---

    async def _persist(self, result: AnalysisResult) -> None:
        await self._write_stationarity(result)
        await self._write_decomposition(result)
        await self._write_forecast(result)
        logger.debug("Persisted results for %s/%s",
result.subcategory, result.metric)

    async def _write_stationarity(self, result: AnalysisResult) ->
None:
        st = result.stationarity
        await self._db.write(
            "ts_stationarity_results",
            [{
                "subcategory":          result.subcategory,
                "metric":               result.metric,
                "adf_statistic":        float(st.adf_statistic),
                "adf_pvalue":           float(st.adf_pvalue),
                "adf_n_lags":           int(st.adf_n_lags),
                "adf_is_stationary":    bool(st.adf_is_stationary),
                "kpss_statistic":       float(st.kpss_statistic),
                "kpss_pvalue":          float(st.kpss_pvalue),
                "kpss_n_lags":          int(st.kpss_n_lags),
                "kpss_is_stationary":   bool(st.kpss_is_stationary),
                "is_stationary":        bool(st.is_stationary),
                "transformations":      st.transformations,
                "recommendation":       st.recommendation,
            }],
        )

    async def _write_decomposition(self, result: AnalysisResult) ->
None:
        dec = result.decomposition
        rows = [
            {
                "subcategory":   result.subcategory,
                "metric":        result.metric,
                "period":        ts.date(),
                "trend":         None if np.isnan(t) else float(t),
```

```
                "seasonal":      None if np.isnan(se) else float(se),
                "residual":      None if np.isnan(r) else float(r),
                "decomp_period": dec.decomp_period,
                "method":        dec.method,
            }
            for ts, t, se, r in zip(
                dec.trend.index,
                dec.trend.values,
                dec.seasonal.values,
                dec.residual.values,
            )
        ]
        await self._db.write("ts_decomposition_results", rows)

    async def _write_forecast(self, result: AnalysisResult) -> None:
        fc = result.forecast
        rows = [
            {
                "subcategory":      result.subcategory,
                "metric":           result.metric,
                "period":           ts.date(),
                "forecast_value":   float(v),
                "conf_int_lower":   float(lo),
                "conf_int_upper":   float(hi),
                "model_name":       fc.model_name,
                "model_order":      fc.model_order,
                "model_aic":        round(fc.aic, 4),
            }
            for ts, v, lo, hi in zip(
                fc.forecast.index,
                fc.forecast.values,
                fc.conf_int_lower.values,
                fc.conf_int_upper.values,
            )
        ]
        await self._db.write(
            "ts_forecast_results",
            rows,
            on_conflict="(subcategory, metric, period, analysed_at)
DO NOTHING",
        )
```

Файл visualizer.py

```
from __future__ import annotations

from pathlib import Path
from typing import Literal

import logging
import matplotlib.dates as mdates
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
from analysis import AnalysisResult, DecompositionResult,
ForecastResult, StationarityResult

logger = logging.getLogger(__name__)

# ── style defaults ──────────────────────────────────────────────

_PALETTE = {
    "observed":  "#2C7BB6",
    "trend":     "#D7191C",
    "seasonal":  "#1A9641",
    "residual":  "#756BB1",
    "forecast":  "#F46D43",
    "ci":        "#FEE08B",
    "rolling7":  "#ABDDA4",
    "rolling30": "#2C7BB6",
    "stationary": "#1A9641",
    "non_stationary": "#D7191C",
    "grid":      "#E8E8E8",
    "neutral":   "#555555",
}

_DATE_FMT = mdates.AutoDateFormatter(mdates.AutoDateLocator())


def _apply_style(ax: plt.Axes, title: str = "", xlabel: str = "",
ylabel: str = "") -> None:
    ax.set_title(title, fontsize=11, fontweight="bold", pad=8)
    ax.set_xlabel(xlabel, fontsize=9)
    ax.set_ylabel(ylabel, fontsize=9)
    ax.grid(True, color=_PALETTE["grid"], linewidth=0.7, linestyle="-
-")
    ax.tick_params(labelsize=8)

ax.xaxis.set_major_formatter(mdates.AutoDateFormatter(mdates.AutoDate
Locator()))
    plt.setp(ax.xaxis.get_majorticklabels(), rotation=30, ha="right")
    ax.spines[["top", "right"]].set_visible(False)


# ── visualiser ──────────────────────────────────────────────────

class TimeSeriesVisualizer:
    """
    Generates matplotlib figures from AnalysisResult objects.

    Usage
    -----
    viz = TimeSeriesVisualizer(output_dir="reports/")

    viz.plot_full_report(result)                    # all panels in
one figure
    viz.plot_series(result)                         # raw series only
    viz.plot_stationarity(result)                   # rolling stats +
test badges
```

```
    viz.plot_decomposition(result)                  # trend /
seasonal / residual
    viz.plot_acf_pacf(result)                       # ACF + PACF
correlograms
    viz.plot_forecast(result)                       # historical +
forecast + CI
    """

    def __init__(
        self,
        output_dir: str | Path | None = None,
        figsize_base: tuple[int, int] = (14, 5),
        dpi: int = 130,
        save_format: Literal["png", "pdf", "svg"] = "png",
    ) -> None:
        self._output_dir = Path(output_dir) if output_dir else None
        self._figsize_base = figsize_base
        self._dpi = dpi
        self._fmt = save_format

        if self._output_dir:
            self._output_dir.mkdir(parents=True, exist_ok=True)

        plt.rcParams.update({
            "font.family": "DejaVu Sans",
            "axes.facecolor": "#FAFAFA",
            "figure.facecolor": "white",
        })

    # ------------------------------------------------------------------
---
    # full report
    # ------------------------------------------------------------------
---

    def plot_full_report(self, result: AnalysisResult) -> plt.Figure:
        """
        Seven-panel figure:
          Row 0 : raw series
          Row 1 : rolling mean/std  |  stationarity badges
          Row 2 : trend             |  seasonal  |  residual
          Row 3 : ACF               |  PACF
          Row 4 : forecast + CI
        """
        fig = plt.figure(figsize=(16, 22), dpi=self._dpi)
        fig.suptitle(
            f"Time Series Analysis – {result.subcategory} /
{result.metric}",
            fontsize=14, fontweight="bold", y=0.99,
        )

        gs = gridspec.GridSpec(5, 2, figure=fig, hspace=0.55,
wspace=0.35)

        # row 0 – raw series (full width)
        ax_series = fig.add_subplot(gs[0, :])
        self._draw_series(ax_series, result.series,
result.transformed_series)
```

```
        # row 1 - rolling stats | stationarity table
        ax_rolling = fig.add_subplot(gs[1, 0])
        ax_stat    = fig.add_subplot(gs[1, 1])
        self._draw_rolling(ax_rolling, result.series)
        self._draw_stationarity_table(ax_stat, result.stationarity)

        # row 2 - decomposition components
        ax_trend    = fig.add_subplot(gs[2, 0])
        ax_seasonal = fig.add_subplot(gs[2, 1])
        ax_residual = fig.add_subplot(gs[3, 0])
        self._draw_component(ax_trend,    result.decomposition.trend,
"Trend",    _PALETTE["trend"])
        self._draw_component(ax_seasonal,
result.decomposition.seasonal, "Seasonal", _PALETTE["seasonal"])
        self._draw_component(ax_residual,
result.decomposition.residual, "Residual", _PALETTE["residual"],
bar=True)

        # row 3 - ACF / PACF (right panel)
        ax_acf  = fig.add_subplot(gs[3, 1])
        self._draw_acf(ax_acf, result.transformed_series,
partial=False)

        # row 4 - forecast (full width)
        ax_fc = fig.add_subplot(gs[4, :])
        self._draw_forecast(ax_fc, result.series, result.forecast)

        self._save(fig, result, "full_report")
        return fig

    # ------------------------------------------------------------------
---
    # individual plots
    # ------------------------------------------------------------------
---

    def plot_series(self, result: AnalysisResult) -> plt.Figure:
        fig, ax = plt.subplots(figsize=self._figsize_base,
dpi=self._dpi)
        self._draw_series(ax, result.series,
result.transformed_series)
        fig.tight_layout()
        self._save(fig, result, "series")
        return fig

    def plot_stationarity(self, result: AnalysisResult) ->
plt.Figure:
        fig, (ax_roll, ax_tbl) = plt.subplots(1, 2, figsize=(14, 5),
dpi=self._dpi)
        self._draw_rolling(ax_roll, result.series)
        self._draw_stationarity_table(ax_tbl, result.stationarity)
        fig.suptitle(f"Stationarity - {result.subcategory} /
{result.metric}", fontweight="bold")
        fig.tight_layout()
        self._save(fig, result, "stationarity")
        return fig
```

```
    def plot_decomposition(self, result: AnalysisResult) ->
plt.Figure:
        dec = result.decomposition
        fig, axes = plt.subplots(4, 1, figsize=(14, 12),
dpi=self._dpi, sharex=True)
        self._draw_component(axes[0], dec.observed, "Observed",
_PALETTE["observed"])
        self._draw_component(axes[1], dec.trend,    "Trend",
_PALETTE["trend"])
        self._draw_component(axes[2], dec.seasonal, "Seasonal",
_PALETTE["seasonal"])
        self._draw_component(axes[3], dec.residual, "Residual",
_PALETTE["residual"], bar=True)
        fig.suptitle(
            f"STL Decomposition (period={dec.decomp_period}) - "
            f"{result.subcategory} / {result.metric}",
            fontweight="bold",
        )
        fig.tight_layout()
        self._save(fig, result, "decomposition")
        return fig

    def plot_acf_pacf(self, result: AnalysisResult) -> plt.Figure:
        fig, (ax_acf, ax_pacf) = plt.subplots(1, 2, figsize=(14, 4),
dpi=self._dpi)
        self._draw_acf(ax_acf,  result.transformed_series,
partial=False)
        self._draw_acf(ax_pacf, result.transformed_series,
partial=True)
        fig.suptitle(
            f"ACF / PACF - {result.subcategory} / {result.metric} "
            f"(transformed: {result.stationarity.transformations or
'none'})",
            fontweight="bold",
        )
        fig.tight_layout()
        self._save(fig, result, "acf_pacf")
        return fig

    def plot_forecast(self, result: AnalysisResult) -> plt.Figure:
        fig, ax = plt.subplots(figsize=(14, 5), dpi=self._dpi)
        self._draw_forecast(ax, result.series, result.forecast)
        fig.tight_layout()
        self._save(fig, result, "forecast")
        return fig

    # ------------------------------------------------------------------
---
    # drawing helpers
    # ------------------------------------------------------------------
---

    @staticmethod
    def _draw_series(
        ax: plt.Axes,
        series: pd.Series,
        transformed: pd.Series,
    ) -> None:
```

```
        ax.plot(series.index, series.values,
                color=_PALETTE["observed"], linewidth=1.2,
label="Observed")
        if not transformed.equals(series):
            ax2 = ax.twinx()
            ax2.plot(transformed.index, transformed.values,
                     color=_PALETTE["forecast"], linewidth=0.9,
                     alpha=0.65, linestyle="--", label="Transformed")
            ax2.set_ylabel("Transformed", fontsize=8,
color=_PALETTE["forecast"])
            ax2.tick_params(labelsize=8, colors=_PALETTE["forecast"])
        _apply_style(ax, title="Raw Time Series", ylabel="Value")
        ax.legend(fontsize=8, loc="upper left")

    @staticmethod
    def _draw_rolling(ax: plt.Axes, series: pd.Series) -> None:
        rolling7  = series.rolling(7)
        rolling30 = series.rolling(30)
        ax.plot(series.index, series.values,
                color=_PALETTE["observed"], alpha=0.35,
linewidth=0.8, label="Observed")
        ax.plot(series.index, rolling7.mean(),
                color=_PALETTE["rolling7"], linewidth=1.3, label="7-
period MA")
        ax.plot(series.index, rolling30.mean(),
                color=_PALETTE["rolling30"], linewidth=1.3,
linestyle="--", label="30-period MA")
        ax.fill_between(
            series.index,
            rolling7.mean() - rolling7.std(),
            rolling7.mean() + rolling7.std(),
            alpha=0.15, color=_PALETTE["rolling7"],
        )
        _apply_style(ax, title="Rolling Mean ± Std (7 & 30 periods)",
ylabel="Value")
        ax.legend(fontsize=8)

    @staticmethod
    def _draw_stationarity_table(ax: plt.Axes, st:
StationarityResult) -> None:
        ax.axis("off")
        ax.set_title("Stationarity Test Results", fontsize=11,
fontweight="bold", pad=8)

        def badge(val: bool) -> str:
            return "✓ Stationary" if val else "✗ Non-stationary"

        def badge_color(val: bool) -> str:
            return _PALETTE["stationary"] if val else
_PALETTE["non_stationary"]

        rows = [
            ("ADF statistic",      f"{st.adf_statistic:.4f}"),
            ("ADF p-value",        f"{st.adf_pvalue:.4f}"),
            ("ADF lags",           str(st.adf_n_lags)),
            ("ADF verdict",        badge(st.adf_is_stationary)),
            ("",                   ""),
            ("KPSS statistic",     f"{st.kpss_statistic:.4f}"),
```

```
            ("KPSS p-value",       f"{st.kpss_pvalue:.4f}"),
            ("KPSS lags",          str(st.kpss_n_lags)),
            ("KPSS verdict",       badge(st.kpss_is_stationary)),
            ("",                   ""),
            ("Transformations",    ", ".join(st.transformations) or
"none"),
            ("Combined verdict",   badge(st.is_stationary)),
        ]

        for i, (label, value) in enumerate(rows):
            y = 1 - i * 0.083
            ax.text(0.02, y, label, transform=ax.transAxes,
                    fontsize=9, color=_PALETTE["neutral"], va="top")
            color = _PALETTE["neutral"]
            if "verdict" in label.lower():
                color = badge_color("Stationary" in value)
            ax.text(0.55, y, value, transform=ax.transAxes,
                    fontsize=9, fontweight="bold", color=color,
va="top")

        # recommendation box
        ax.text(
            0.0, -0.05, f"⚑  {st.recommendation}",
            transform=ax.transAxes, fontsize=8, color="#555",
            wrap=True, va="top",
            bbox=dict(boxstyle="round,pad=0.4", facecolor="#FFF9C4",
edgecolor="#F0C040", alpha=0.9),
        )

    @staticmethod
    def _draw_component(
        ax: plt.Axes,
        component: pd.Series,
        label: str,
        color: str,
        bar: bool = False,
    ) -> None:
        if bar:
            ax.bar(component.index, component.values, color=color,
alpha=0.55, width=1)
        else:
            ax.plot(component.index, component.values, color=color,
linewidth=1.2)
        _apply_style(ax, title=label, ylabel="Value")

    @staticmethod
    def _draw_acf(ax: plt.Axes, series: pd.Series, partial: bool =
False) -> None:
        lags = min(40, len(series) // 2 - 1)
        fn = plot_pacf if partial else plot_acf
        fn(series.dropna(), ax=ax, lags=lags, alpha=0.05,
           color=_PALETTE["observed"], vlines_kwargs={"colors":
_PALETTE["observed"]})
        title = "Partial ACF (PACF)" if partial else "Autocorrelation
(ACF)"
        _apply_style(ax, title=title, xlabel="Lag",
ylabel="Correlation")
```

```
    @staticmethod
    def _draw_forecast(
        ax: plt.Axes,
        series: pd.Series,
        fc: ForecastResult,
    ) -> None:
        # historical (last 2× forecast horizon for readability)
        n_show = min(len(series), len(fc.forecast) * 2)
        hist = series.iloc[-n_show:]
        ax.plot(hist.index, hist.values,
                color=_PALETTE["observed"], linewidth=1.2,
label="Historical")

        ax.plot(fc.forecast.index, fc.forecast.values,
                color=_PALETTE["forecast"], linewidth=1.5,
                linestyle="--", marker="o", markersize=3,
label="Forecast")
        ax.fill_between(
            fc.forecast.index,
            fc.conf_int_lower.values,
            fc.conf_int_upper.values,
            alpha=0.25, color=_PALETTE["ci"], label="95 % CI",
        )

        # vertical divider between history and forecast
        ax.axvline(series.index[-1], color="#999", linewidth=0.8,
linestyle=":")

        model_label = f"{fc.model_order}  AIC={fc.aic:.1f}"
        _apply_style(ax, title=f"Forecast ({model_label})",
ylabel="Value")
        ax.legend(fontsize=8)

    # ------------------------------------------------------------------
---
    # save
    # ------------------------------------------------------------------
---

    def _save(self, fig: plt.Figure, result: AnalysisResult, suffix:
str) -> None:
        if not self._output_dir:
            return
        name =
f"{result.subcategory}_{result.metric}_{suffix}.{self._fmt}"
        path = self._output_dir / name.replace(" ", "_").lower()
        fig.savefig(path, bbox_inches="tight", dpi=self._dpi)
        plt.close(fig)
        logger.info("Saved: %s", path)
        return path
```

Файл run_analysis.py

```
"""
run_analysis.py
---------------
```

```
Discovers all product subcategories in the DB, builds their
popularity time
series, identifies the most popular ones, runs full time series
analysis and
forecasting, then generates visualisation reports.

Usage
-----
# full pipeline: populate metrics -> analyse -> visualise
python run_analysis.py

# skip re-populating popularity_metrics (data already loaded)
python run_analysis.py --skip-populate

# filter to top-N subcategories by mean popularity before analysing
python run_analysis.py --top 20

# set minimum number of data points required to include a subcategory
python run_analysis.py --min-points 30

# only show plots interactively, don't save
python run_analysis.py --no-save

# override forecast horizon
python run_analysis.py --forecast-periods 60
"""

import argparse
import asyncio
import logging
import sys
from dataclasses import dataclass

import matplotlib.pyplot as plt

from analysis import AnalysisResult, TimeSeriesAnalyzer
from config import DBConfig
from database import Database
from visualizer import TimeSeriesVisualizer

# ── logging
─────────────────────────────────────────────────────────────

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(name)s - %(message)s",
    handlers=[
        logging.StreamHandler(sys.stdout),
        logging.FileHandler("analysis.log", encoding="utf-8"),
    ],
)
logger = logging.getLogger(__name__)

# ── constants
─────────────────────────────────────────────────────────────

METRIC = "product_count"        # what we measure as "popularity"
```

```
MIN_POINTS_DEFAULT = 14         # minimum time-series length to
analyse
POPULAR_PERCENTILE = 0.5        # subcategories above the 50th
percentile of mean value


# ── SQL
──────────────────────────────────────────────────────────────────
──

# Aggregate distinct product groups per subcategory per calendar day
and upsert
# into popularity_metrics so the time series is always up to date.
SQL_POPULATE_METRICS = """
INSERT INTO popularity_metrics (subcategory, metric, period, value)
SELECT
    split_part(product_type, '>',
array_length(string_to_array(product_type, '>'), 1))
        AS subcategory,
    $1                                      AS metric,
    DATE(timestamp)                         AS period,
    COUNT(DISTINCT item_group_id)::NUMERIC  AS value
FROM products
WHERE
    product_type IS NOT NULL
    AND product_type <> ''
GROUP BY subcategory, period
ON CONFLICT (subcategory, metric, period)
DO UPDATE SET value = EXCLUDED.value;
"""

# Return every subcategory that has at least $2 data points, richest
first.
SQL_LOAD_SUBCATEGORIES = """
SELECT
    subcategory,
    COUNT(*)        AS data_points,
    AVG(value)      AS mean_value,
    MAX(value)      AS max_value,
    MIN(period)     AS series_start,
    MAX(period)     AS series_end
FROM popularity_metrics
WHERE metric = $1
GROUP BY subcategory
HAVING COUNT(*) >= $2
ORDER BY mean_value DESC;
"""


# ── popularity identifier
──────────────────────────────────────────────────────


@dataclass
class SubcategoryProfile:
    subcategory:   str
    data_points:   int
    mean_value:    float
    max_value:     float
```

```
    series_start:  str
    series_end:    str
    is_popular:    bool = False
    popularity_rank: int = 0


def identify_popular(
    profiles: list[SubcategoryProfile],
    top_n: int | None = None,
    percentile: float = POPULAR_PERCENTILE,
) -> list[SubcategoryProfile]:
    """
    Mark each subcategory as popular.

    If top_n is given  → top-N by mean value.
    Otherwise          → mean value above the given percentile of all
subcategories.
    """
    if not profiles:
        return profiles

    ranked = sorted(profiles, key=lambda p: p.mean_value,
reverse=True)
    means  = [p.mean_value for p in ranked]
    threshold = means[int(len(means) * percentile)]

    for rank, profile in enumerate(ranked, start=1):
        profile.popularity_rank = rank
        profile.is_popular = (rank <= top_n) if top_n else
(profile.mean_value >= threshold)

    n_popular = sum(1 for p in profiles if p.is_popular)
    logger.info(
        "Identified %d / %d subcategories as popular
(threshold=%.1f)",
        n_popular, len(profiles), threshold,
    )
    return profiles


# ── console output helpers
─────────────────────────────────────────────────────


_RESET = "\033[0m"

def _color(text: str, code: str) -> str:
    return f"\033[{code}m{text}{_RESET}"


def _banner(text: str, color: str = "96") -> None:
    bar = "═" * 64
    print(f"\n{_color(bar, color)}")
    print(f"{_color(f'  {text}', color)}")
    print(f"{_color(bar, color)}")


def _print_subcategory_table(profiles: list[SubcategoryProfile]) ->
None:
```

```
    header = f"  {'#':<5} {'Subcategory':<28} {'Points':>7}
{'Mean':>9} {'Max':>9}"
    divider = f"  {'─'*5} {'─'*28} {'─'*7} {'─'*9} {'─'*9}"

    popular = [p for p in profiles if p.is_popular]
    other   = [p for p in profiles if not p.is_popular]

    _banner(f"POPULAR SUBCATEGORIES  ({len(popular)})", "92")
    print(header)
    print(divider)
    for p in popular:
        print(
            f"  {p.popularity_rank:<5} {p.subcategory:<28} "
            f"{p.data_points:>7} {p.mean_value:>9.1f}
{p.max_value:>9.1f}"
        )

    _banner(f"OTHER SUBCATEGORIES  ({len(other)}  - below
threshold)", "33")
    print(header)
    print(divider)
    for p in other:
        print(
            f"  {p.popularity_rank:<5} {p.subcategory:<28} "
            f"{p.data_points:>7} {p.mean_value:>9.1f}
{p.max_value:>9.1f}"
        )


def _print_result_summary(result: AnalysisResult) -> None:
    st  = result.stationarity
    fc  = result.forecast
    dec = result.decomposition
    verdict = (
        _color("✓ stationary",     "92") if st.is_stationary
        else _color("✗ non-stationary", "91")
    )
    print(
        f"\n  {_color(result.subcategory, '1')}\n"
        f"  {'─' * 42}\n"
        f"  Length        : {len(result.series)} points  "
        f"({result.series.index[0].date()} → {result.series.index[-
1].date()})\n"
        f"  Transforms    : {st.transformations or ['none']}\n"
        f"  Stationarity  : {verdict}  "
        f"ADF p={st.adf_pvalue:.4f}  KPSS p={st.kpss_pvalue:.4f}\n"
        f"  Decomposition : {dec.method},
period={dec.decomp_period}\n"
        f"  Model         : {fc.model_order}  AIC={fc.aic:.2f}\n"
        f"  Forecast      : {len(fc.forecast)} periods ahead\n"
        f"  Note          : {st.recommendation}\n"
    )


def _print_forecast_summary(results: list[AnalysisResult]) -> None:
    _banner("FORECAST SUMMARY - predicted mean vs last observed
value", "96")
    print(
```

```
        f"  {'Subcategory':<28} {'Last observed':>14} "
        f"{'Forecast mean':>14} {'Change':>10}"
    )
    print(f"  {'─'*28} {'─'*14} {'─'*14} {'─'*10}")

    for r in sorted(results, key=lambda x:
x.forecast.forecast.mean(), reverse=True):
        last_val     = r.series.iloc[-1]
        forecast_avg = r.forecast.forecast.mean()
        delta        = forecast_avg - last_val
        arrow        = "↑" if delta >= 0 else "↓"
        change_str   = f"{arrow} {abs(delta):.1f}"
        change_color = "92" if delta >= 0 else "91"
        print(
            f"  {r.subcategory:<28} {last_val:>14.1f} "
            f"{forecast_avg:>14.1f} "
            f"{_color(f'{change_str:>10}', change_color)}"
        )


def print_raw_stats(result: AnalysisResult) -> None:
    """Print every number behind the visualisation for one
AnalysisResult."""
    st  = result.stationarity
    dec = result.decomposition
    fc  = result.forecast

    _banner(f"RAW STATS – {result.subcategory} / {result.metric}",
"95")

    # ── series overview
────────────────────────────────────────────────
    s = result.series
    print(f"\n  {'─'*20} SERIES OVERVIEW {'─'*20}")
    print(f"  Length          : {len(s)}")
    print(f"  From / To       : {s.index[0].date()}  →  {s.index[-
1].date()}")
    print(f"  Mean            : {s.mean():.4f}")
    print(f"  Std dev         : {s.std():.4f}")
    print(f"  Min / Max       : {s.min():.4f}  /  {s.max():.4f}")
    print(f"  Median          : {s.median():.4f}")
    print(f"  Coeff. of var.  : {s.std() / s.mean():.4f}")
    print(f"  Transformations : {st.transformations or ['none']}")

    # ── stationarity
──────────────────────────────────────────────────
    print(f"\n  {'─'*20} STATIONARITY TESTS {'─'*17}")
    print(f"  {'':30} {'ADF':>12}  {'KPSS':>12}")
    print(f"  {'─'*30} {'─'*12}  {'─'*12}")
    print(f"  {'Statistic':<30} {st.adf_statistic:>12.6f}
{st.kpss_statistic:>12.6f}")
    print(f"  {'p-value':<30} {st.adf_pvalue:>12.6f}
{st.kpss_pvalue:>12.6f}")
    print(f"  {'Lags used':<30} {st.adf_n_lags:>12}
{st.kpss_n_lags:>12}")
    adf_v  = _color("✓ stationary",     "92") if st.adf_is_stationary
else _color("✗ non-stationary", "91")
```

```
    kpss_v = _color("✓ stationary",     "92") if
st.kpss_is_stationary else _color("✗ non-stationary", "91")
    print(f"  {'Verdict (p<0.05 / p>0.05)':<30} {adf_v:>12}
{kpss_v:>12}")
    combined = _color("✓ stationary", "92") if st.is_stationary else
_color("✗ non-stationary", "91")
    print(f"  {'Combined verdict':<30} {combined}")
    print(f"  Recommendation  : {st.recommendation}")

    # ── decomposition
──────────────────────────────────────────────
    print(f"\n  {'─'*20} STL DECOMPOSITION
(period={dec.decomp_period}) {'─'*5}")
    print(f"  {'Component':<12} {'Mean':>10} {'Std':>10} {'Min':>10}
{'Max':>10}")
    print(f"  {'─'*12} {'─'*10} {'─'*10} {'─'*10} {'─'*10}")
    for label, component in [
        ("Observed",  dec.observed),
        ("Trend",     dec.trend),
        ("Seasonal",  dec.seasonal),
        ("Residual",  dec.residual),
    ]:
        c = component.dropna()
        print(
            f"  {label:<12} {c.mean():>10.4f} {c.std():>10.4f} "
            f"{c.min():>10.4f} {c.max():>10.4f}"
        )
    strength_t = max(0, 1 - dec.residual.var() / (dec.trend   +
dec.residual).var())
    strength_s = max(0, 1 - dec.residual.var() / (dec.seasonal +
dec.residual).var())
    print(f"\n  Trend strength    : {strength_t:.4f}  (0=none →
1=strong)")
    print(f"  Seasonal strength : {strength_s:.4f}  (0=none →
1=strong)")

    # ── forecast
──────────────────────────────────────────────────
    print(f"\n  {'─'*20} FORECAST {'─'*27}")
    print(f"  Model           : {fc.model_order}")
    print(f"  AIC             : {fc.aic:.4f}")
    print(f"  Horizon         : {len(fc.forecast)} periods")
    print(f"  {'Date':<14} {'Forecast':>12} {'CI lower':>12} {'CI
upper':>12}")
    print(f"  {'─'*14} {'─'*12} {'─'*12} {'─'*12}")
    for date, val, lo, hi in zip(
        fc.forecast.index,
        fc.forecast.values,
        fc.conf_int_lower.values,
        fc.conf_int_upper.values,
    ):
        print(f"  {str(date.date()):<14} {val:>12.4f} {lo:>12.4f}
{hi:>12.4f}")
    print()
```

```
# ── main pipeline ─────────────────────────────────────────────────────────

async def run(args: argparse.Namespace) -> None:
    db_cfg = DBConfig.from_env()

    async with Database(dsn=db_cfg.dsn, min_size=db_cfg.min_pool,
max_size=db_cfg.max_pool) as db:
        await db.initialise()

        # ── step 1: populate time series from products
────────────────
        if not args.skip_populate:
            logger.info("Populating popularity_metrics from products
table…")
            await db.execute(SQL_POPULATE_METRICS, METRIC)
            logger.info("popularity_metrics table updated.")

        # ── step 2: load all subcategories with enough data
──────────
        rows = await db.execute_fetch(SQL_LOAD_SUBCATEGORIES, METRIC,
args.min_points)
        if not rows:
            logger.error(
                "No subcategories found with >= %d data points. "
                "Try a lower --min-points, or remove --skip-
populate.",
                args.min_points,
            )
            return

        logger.info("Found %d subcategories with sufficient data.",
len(rows))

        profiles = [
            SubcategoryProfile(
                subcategory  = r["subcategory"],
                data_points  = int(r["data_points"]),
                mean_value   = float(r["mean_value"]),
                max_value    = float(r["max_value"]),
                series_start = str(r["series_start"]),
                series_end   = str(r["series_end"]),
            )
            for r in rows
        ]

        # ── step 3: identify popular subcategories
─────────────────
        profiles = identify_popular(profiles, top_n=args.top)
        _print_subcategory_table(profiles)

        popular = [p for p in profiles if p.is_popular]
        if not popular:
            logger.warning("No popular subcategories to analyse.")
            return

        # ── step 4: run analysis
──────────────────────────────────
```

```
        analyser = TimeSeriesAnalyzer(db)
        pairs    = [(p.subcategory, METRIC) for p in popular]

        logger.info("Analysing %d popular subcategories…",
len(pairs))
        results = await analyser.run_many(
            pairs,
            forecast_periods=args.forecast_periods,
            seasonal_period=args.seasonal_period,
            refresh_views=True,
        )

        if not results:
            logger.error("Analysis returned no results.")
            return

        # ── step 5: visualise
────────────────────────────────────────
        viz = TimeSeriesVisualizer(
            output_dir=None if args.no_save else args.output_dir,
            save_format="png",
        )
        plot_method_map = {
            "full":          "plot_full_report",
            "series":        "plot_series",
            "stationarity":  "plot_stationarity",
            "decomposition": "plot_decomposition",
            "acf-pacf":      "plot_acf_pacf",
            "forecast":      "plot_forecast",
        }
        plot_method = plot_method_map[args.plot]

        _banner(f"ANALYSIS RESULTS  ({len(results)} subcategories)",
"94")
        for result in results:
            _print_result_summary(result)
            if args.stats:
                print_raw_stats(result)
            try:
                fig = getattr(viz, plot_method)(result)
                if args.no_save:
                    plt.show()
                else:
                    plt.close(fig)
            except Exception as exc:
                logger.error("Visualisation failed for %s: %s",
result.subcategory, exc)

        # ── step 6: final forecast table
──────────────────────────
        _print_forecast_summary(results)


# ── CLI
──────────────────────────────────────────────────────────────────────
──

def parse_args() -> argparse.Namespace:
```

```
    p = argparse.ArgumentParser(
        description="Identify popular clothing subcategories and
forecast their popularity."
    )
    p.add_argument(
        "--skip-populate", action="store_true",
        help="Skip re-populating popularity_metrics (use existing
data).",
    )
    p.add_argument(
        "--top", type=int, default=None,
        help="Limit analysis to top-N subcategories by mean
popularity.",
    )
    p.add_argument(
        "--min-points", type=int, default=MIN_POINTS_DEFAULT,
        help=f"Minimum series length to include a subcategory
(default: {MIN_POINTS_DEFAULT}).",
    )
    p.add_argument(
        "--forecast-periods", type=int, default=30,
        help="Number of future periods to forecast (default: 30).",
    )
    p.add_argument(
        "--seasonal-period", type=int, default=None,
        help="Override auto-detected seasonal period (e.g. 7, 12,
52).",
    )
    p.add_argument(
        "--output-dir", type=str, default="reports",
        help="Directory for saved plot images (default: reports/).",
    )
    p.add_argument(
        "--no-save", action="store_true",
        help="Show plots interactively instead of saving to disk.",
    )
    p.add_argument(
        "--stats", action="store_true",
        help="Print full raw statistics (stationarity, decomposition,
forecast table) to console.",
    )
    p.add_argument(
        "--plot", type=str, default="full",
        choices=["full", "series", "stationarity", "decomposition",
"acf-pacf", "forecast"],
        help="Which plot type to generate per subcategory (default:
full).",
    )
    return p.parse_args()


if __name__ == "__main__":
    asyncio.run(run(parse_args()))
```

Мова програмування: SQL.

База даних: PostgreSQL 17.

Файл setup.sql

```
    --
================================================================
========
    -- setup.sql
    -- Run once to bootstrap the database.
    -- Safe to re-run: all statements use IF NOT EXISTS / DO NOTHING
guards.
    --
================================================================
========


    -- ------------------------------------------------------------
---------------
    -- Extensions
    -- ------------------------------------------------------------
---------------

    CREATE EXTENSION IF NOT EXISTS "uuid-ossp";      --
uuid_generate_v4()
    CREATE EXTENSION IF NOT EXISTS "pg_trgm";        -- trigram
indexes for LIKE / ILIKE


    -- ------------------------------------------------------------
---------------
    -- products
    -- Central table - one row per SKU / variant.
    -- ------------------------------------------------------------
---------------

    CREATE TABLE IF NOT EXISTS products (
        -- identity
        id                  TEXT            NOT NULL,
        item_group_id       TEXT            NOT NULL,
        mpn                 TEXT,
        gtin                TEXT,

        -- content
        title               TEXT            NOT NULL,
        description         TEXT,
        brand               TEXT,

        -- media
        image_link          TEXT,
        additional_image_link  TEXT[],       -- array of URLs

        -- classification
        link                TEXT            NOT NULL,
        gender              TEXT            CHECK (gender IN ('male',
'female')),
        age_group           TEXT            DEFAULT 'adult',
```

```
        color               TEXT,
        size                TEXT,
        product_type        TEXT,           -- breadcrumb path, e.g.
"Women>Dresses>Midi"
        condition           TEXT            CHECK (condition IN
('new', 'used', 'refurbished')),

        -- pricing
        availability        TEXT            CHECK (availability IN
('in_stock', 'out_of_stock')),
        price               TEXT,           -- stored as "123.0 GBP"
        sale_price          TEXT,

        -- ratings
        rating_value        NUMERIC(3, 1),
        review_count        INTEGER,
        best_rating         NUMERIC(3, 1),
        worst_rating        NUMERIC(3, 1),

        -- meta
        timestamp           TIMESTAMP       NOT NULL DEFAULT NOW(),

        CONSTRAINT products_pkey PRIMARY KEY (id)
    );

    -- indexes
    CREATE INDEX IF NOT EXISTS idx_products_item_group_id  ON
products (item_group_id);
    CREATE INDEX IF NOT EXISTS idx_products_brand          ON
products (brand);
    CREATE INDEX IF NOT EXISTS idx_products_gender         ON
products (gender);
    CREATE INDEX IF NOT EXISTS idx_products_availability   ON
products (availability);
    CREATE INDEX IF NOT EXISTS idx_products_timestamp      ON
products (timestamp DESC);
    CREATE INDEX IF NOT EXISTS idx_products_title_trgm     ON
products USING gin (title gin_trgm_ops);


    -- ------------------------------------------------------------
---------------
    -- scrape_logs
    -- One row per log entry emitted by the parser / scraper.
    -- ------------------------------------------------------------
---------------

    CREATE TABLE IF NOT EXISTS scrape_logs (
        id              BIGSERIAL       PRIMARY KEY,
        log_type        TEXT            NOT NULL CHECK (log_type IN
('success', 'error')),
        description     TEXT            NOT NULL,
        link            TEXT            NOT NULL,
        additional_data TEXT,
        timestamp       TIMESTAMP       NOT NULL DEFAULT NOW()
    );
```

```
    CREATE INDEX IF NOT EXISTS idx_scrape_logs_log_type   ON
scrape_logs (log_type);
    CREATE INDEX IF NOT EXISTS idx_scrape_logs_link        ON
scrape_logs (link);
    CREATE INDEX IF NOT EXISTS idx_scrape_logs_timestamp   ON
scrape_logs (timestamp DESC);


    -- ------------------------------------------------------------
--------------
    -- listing_pages
    -- Tracks which listing / category URLs have been crawled and
their state.
    -- Useful for resuming interrupted crawls and avoiding duplicate
work.
    -- ------------------------------------------------------------
--------------

    CREATE TABLE IF NOT EXISTS listing_pages (
        id              BIGSERIAL       PRIMARY KEY,
        url             TEXT            NOT NULL UNIQUE,
        status          TEXT            NOT NULL DEFAULT 'pending'
                                        CHECK (status IN ('pending',
'in_progress', 'done', 'failed')),
        total_products  INTEGER,
        products_found  INTEGER,
        current_page    INTEGER,
        error           TEXT,
        scraped_at      TIMESTAMP,
        created_at      TIMESTAMP       NOT NULL DEFAULT NOW()
    );

    CREATE INDEX IF NOT EXISTS idx_listing_pages_status      ON
listing_pages (status);
    CREATE INDEX IF NOT EXISTS idx_listing_pages_scraped_at  ON
listing_pages (scraped_at DESC);


    --
============================================================================
========
    -- TIME SERIES ANALYSIS SCHEMA
    --
============================================================================
========


    -- ------------------------------------------------------------
--------------
    -- popularity_metrics
    -- Raw time series data - one row per (subcategory, metric,
period).
    -- This is the source table for all materialized views below.
    -- ------------------------------------------------------------
--------------

    CREATE TABLE IF NOT EXISTS popularity_metrics (
        id              BIGSERIAL       PRIMARY KEY,
```

```
        subcategory     TEXT            NOT NULL,
        metric          TEXT            NOT NULL,   -- e.g. 'views',
'sales', 'popularity_index'
        period          DATE            NOT NULL,
        value           NUMERIC         NOT NULL,
        created_at      TIMESTAMP       NOT NULL DEFAULT NOW(),

        CONSTRAINT popularity_metrics_uq UNIQUE (subcategory, metric,
period)
    );

    CREATE INDEX IF NOT EXISTS idx_pop_metrics_subcategory  ON
popularity_metrics (subcategory);
    CREATE INDEX IF NOT EXISTS idx_pop_metrics_metric       ON
popularity_metrics (metric);
    CREATE INDEX IF NOT EXISTS idx_pop_metrics_period       ON
popularity_metrics (period DESC);
    CREATE INDEX IF NOT EXISTS idx_pop_metrics_lookup       ON
popularity_metrics (subcategory, metric, period DESC);


    -- ------------------------------------------------------------
---------------
    -- ts_stationarity_results
    -- ADF + KPSS test results produced by the Python analyser.
    -- ------------------------------------------------------------
---------------

    CREATE TABLE IF NOT EXISTS ts_stationarity_results (
        id                  BIGSERIAL   PRIMARY KEY,
        subcategory         TEXT        NOT NULL,
        metric              TEXT        NOT NULL,
        analysed_at         TIMESTAMP   NOT NULL DEFAULT NOW(),

        -- ADF test
        adf_statistic       NUMERIC,
        adf_pvalue          NUMERIC,
        adf_n_lags          INTEGER,
        adf_is_stationary   BOOLEAN,

        -- KPSS test
        kpss_statistic      NUMERIC,
        kpss_pvalue         NUMERIC,
        kpss_n_lags         INTEGER,
        kpss_is_stationary  BOOLEAN,

        -- combined verdict
        is_stationary       BOOLEAN,
        transformations     TEXT[],     -- e.g. ARRAY['log',
'diff_1']
        recommendation      TEXT
    );

    CREATE INDEX IF NOT EXISTS idx_ts_stat_subcategory  ON
ts_stationarity_results (subcategory, metric);
    CREATE INDEX IF NOT EXISTS idx_ts_stat_analysed_at  ON
ts_stationarity_results (analysed_at DESC);
```

```
    -- ------------------------------------------------------------
---------------
    -- ts_decomposition_results
    -- One row per (subcategory, metric, period) for trend / seasonal
/ residual.
    -- ------------------------------------------------------------
---------------

    CREATE TABLE IF NOT EXISTS ts_decomposition_results (
        id              BIGSERIAL   PRIMARY KEY,
        subcategory     TEXT        NOT NULL,
        metric          TEXT        NOT NULL,
        period          DATE        NOT NULL,
        analysed_at     TIMESTAMP   NOT NULL DEFAULT NOW(),

        trend           NUMERIC,
        seasonal        NUMERIC,
        residual        NUMERIC,
        decomp_period   INTEGER,        -- detected seasonal period
(e.g. 7, 30, 365)
        method          TEXT            -- 'STL' | 'classical'
    );

    CREATE INDEX IF NOT EXISTS idx_ts_decomp_lookup      ON
ts_decomposition_results (subcategory, metric, period DESC);
    CREATE INDEX IF NOT EXISTS idx_ts_decomp_analysed_at ON
ts_decomposition_results (analysed_at DESC);


    -- ------------------------------------------------------------
---------------
    -- ts_forecast_results
    -- One row per (subcategory, metric, future period).
    -- ------------------------------------------------------------
---------------

    CREATE TABLE IF NOT EXISTS ts_forecast_results (
        id                  BIGSERIAL   PRIMARY KEY,
        subcategory         TEXT        NOT NULL,
        metric              TEXT        NOT NULL,
        period              DATE        NOT NULL,   -- forecast date
        analysed_at         TIMESTAMP   NOT NULL DEFAULT NOW(),

        forecast_value      NUMERIC     NOT NULL,
        conf_int_lower      NUMERIC,
        conf_int_upper      NUMERIC,

        -- model metadata
        model_name          TEXT,       -- e.g. 'SARIMA'
        model_order         TEXT,       -- e.g. '(1,1,1)(1,1,1,7)'
        model_aic           NUMERIC,

        CONSTRAINT ts_forecast_uq UNIQUE (subcategory, metric,
period, analysed_at)
    );
```

```
    CREATE INDEX IF NOT EXISTS idx_ts_forecast_lookup     ON
ts_forecast_results (subcategory, metric, period DESC);
    CREATE INDEX IF NOT EXISTS idx_ts_forecast_analysed   ON
ts_forecast_results (analysed_at DESC);


    --
================================================================================
    -- MATERIALIZED VIEWS
    -- Refresh with: SELECT refresh_ts_views();
    -- Or individually: REFRESH MATERIALIZED VIEW CONCURRENTLY
mv_<name>;
    --
================================================================================


    -- ------------------------------------------------------------------
---------------
    -- mv_popularity_summary
    -- Latest descriptive statistics per (subcategory, metric).
    -- ------------------------------------------------------------------
---------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_popularity_summary AS
    SELECT
        subcategory,
        metric,
        COUNT(*)                                AS data_points,
        MIN(period)                             AS series_start,
        MAX(period)                             AS series_end,
        ROUND(AVG(value)::NUMERIC, 4)           AS mean_value,
        ROUND(STDDEV(value)::NUMERIC, 4)        AS stddev_value,
        MIN(value)                              AS min_value,
        MAX(value)                              AS max_value,
        ROUND(
            PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY
value)::NUMERIC, 4
        )                                       AS median_value,
        -- coefficient of variation - higher = more volatile
        ROUND(
            (STDDEV(value) / NULLIF(AVG(value), 0))::NUMERIC, 4
        )                                       AS cv
    FROM popularity_metrics
    GROUP BY subcategory, metric
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_pop_summary
        ON mv_popularity_summary (subcategory, metric);


    -- ------------------------------------------------------------------
---------------
    -- mv_popularity_rolling
    -- 7-period and 30-period rolling mean + stddev per data point.
    -- Useful for spotting local trends and volatility shifts.
```

```
    -- ------------------------------------------------------------
--------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_popularity_rolling AS
    SELECT
        subcategory,
        metric,
        period,
        value,
        ROUND(AVG(value) OVER w7::NUMERIC,  4)    AS rolling_mean_7,
        ROUND(STDDEV(value) OVER w7::NUMERIC, 4)  AS rolling_std_7,
        ROUND(AVG(value) OVER w30::NUMERIC,  4)   AS rolling_mean_30,
        ROUND(STDDEV(value) OVER w30::NUMERIC, 4) AS rolling_std_30
    FROM popularity_metrics
    WINDOW
        w7  AS (PARTITION BY subcategory, metric ORDER BY period
                ROWS BETWEEN 6 PRECEDING AND CURRENT ROW),
        w30 AS (PARTITION BY subcategory, metric ORDER BY period
                ROWS BETWEEN 29 PRECEDING AND CURRENT ROW)
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_pop_rolling
        ON mv_popularity_rolling (subcategory, metric, period);


    -- ------------------------------------------------------------
--------------
    -- mv_popularity_yoy
    -- Year-over-year change per data point (requires ≥ 1 year of
history).
    -- ------------------------------------------------------------
--------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_popularity_yoy AS
    SELECT
        cur.subcategory,
        cur.metric,
        cur.period,
        cur.value
AS current_value,
        prev.value
AS prev_year_value,
        ROUND((cur.value - prev.value)::NUMERIC, 4)
AS yoy_abs_change,
        ROUND(
            ((cur.value - prev.value) / NULLIF(prev.value, 0) *
100)::NUMERIC, 2
        )
AS yoy_pct_change
    FROM popularity_metrics cur
    LEFT JOIN popularity_metrics prev
        ON prev.subcategory = cur.subcategory
        AND prev.metric      = cur.metric
        AND prev.period      = cur.period - INTERVAL '1 year'
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_pop_yoy
        ON mv_popularity_yoy (subcategory, metric, period);
```

```
    -- ------------------------------------------------------------
---------------
    -- mv_latest_stationarity
    -- Most recent stationarity test result per (subcategory,
metric).
    -- ------------------------------------------------------------
---------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_latest_stationarity AS
    SELECT DISTINCT ON (subcategory, metric)
        subcategory,
        metric,
        analysed_at,
        adf_statistic,
        adf_pvalue,
        adf_is_stationary,
        kpss_statistic,
        kpss_pvalue,
        kpss_is_stationary,
        is_stationary,
        transformations,
        recommendation
    FROM ts_stationarity_results
    ORDER BY subcategory, metric, analysed_at DESC
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_latest_stat
        ON mv_latest_stationarity (subcategory, metric);


    -- ------------------------------------------------------------
---------------
    -- mv_latest_forecast
    -- Most recent forecast horizon per (subcategory, metric).
    -- ------------------------------------------------------------
---------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_latest_forecast AS
    SELECT DISTINCT ON (subcategory, metric, period)
        subcategory,
        metric,
        period,
        forecast_value,
        conf_int_lower,
        conf_int_upper,
        model_name,
        model_order,
        model_aic,
        analysed_at
    FROM ts_forecast_results
    ORDER BY subcategory, metric, period, analysed_at DESC
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_latest_forecast
        ON mv_latest_forecast (subcategory, metric, period);
```

```
    -- ------------------------------------------------------------
---------------
    -- mv_latest_decomposition
    -- Most recent decomposition snapshot per (subcategory, metric,
period).
    -- ------------------------------------------------------------
---------------

    CREATE MATERIALIZED VIEW IF NOT EXISTS mv_latest_decomposition AS
    SELECT DISTINCT ON (subcategory, metric, period)
        subcategory,
        metric,
        period,
        trend,
        seasonal,
        residual,
        decomp_period,
        method,
        analysed_at
    FROM ts_decomposition_results
    ORDER BY subcategory, metric, period, analysed_at DESC
    WITH DATA;

    CREATE UNIQUE INDEX IF NOT EXISTS uidx_mv_latest_decomp
        ON mv_latest_decomposition (subcategory, metric, period);


    --
================================================================
========
    -- HELPER FUNCTION - refresh all materialized views in dependency
order
    -- Usage: SELECT refresh_ts_views();
    --
================================================================
========

    CREATE OR REPLACE FUNCTION refresh_ts_views()
    RETURNS void
    LANGUAGE plpgsql AS $$
    BEGIN
        REFRESH MATERIALIZED VIEW CONCURRENTLY mv_popularity_summary;
        REFRESH MATERIALIZED VIEW CONCURRENTLY mv_popularity_rolling;
        REFRESH MATERIALIZED VIEW CONCURRENTLY mv_popularity_yoy;
        REFRESH MATERIALIZED VIEW CONCURRENTLY
mv_latest_stationarity;
        REFRESH MATERIALIZED VIEW CONCURRENTLY mv_latest_forecast;
        REFRESH MATERIALIZED VIEW CONCURRENTLY
mv_latest_decomposition;
    END;
    $$;
```

Додаткові файли

Файл requirements.txt

```
aiohttp>=3.9.0
asyncpg>=0.29.0
beautifulsoup4>=4.12.0
pandas>=2.0.0
python-dotenv>=1.0.0
brotli==1.2.0
scipy>=1.12.0
statsmodels>=0.14.0
matplotlib>=3.8.0
```