МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

НАЗАРУК ЯРОСЛАВ СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА BI - ДОДАТКУ ДЛЯ АНАЛІЗУ ПРОДАЖІВ У РОЗДРІБНІЙ ТОРГІВЛІ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Тетяна БАЦЕНКО, доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Назарук Я. С. Розробка BI-додатку для аналізу продажів у роздрібній торгівлі. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано принципи побудови систем бізнес-аналітики для роздрібної торгівлі. За допомогою технологій PostgreSQL, Python (pandas, SQLAlchemy, Prophet) та Microsoft Power BI розроблено BI-додаток, основна мета якого – автоматизація збору, очищення та візуалізації даних продажів, прогнозування попиту та сегментація клієнтів для підтримки управлінських рішень.

Ключові слова: бізнес-аналітика, роздрібна торгівля, ETL-конвеєр, прогнозування, візуалізація даних, Power BI, PostgreSQL.

54 ст., 16 рис., 2 табл., 3 дод., 23 джерел.

ABSTRACT

Nazaruk Y. S. Development of a BI Application for Retail Sales Analytics. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

In the qualification (bachelor's) work the principles of building business intelligence systems for retail are researched and analyzed. Using PostgreSQL, Python (pandas, SQLAlchemy, Prophet) and Microsoft Power BI, a BI application was developed, the main purpose of which is to automate data collection, cleaning and visualization of sales data, demand forecasting and customer segmentation to support management decisions.

Keywords: business intelligence, retail sales, ETL pipeline, forecasting, data visualization, Power BI, PostgreSQL.

54 p., 16 fig., 2 tbl., 3 app., 23 ref.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

BI (Business Intelligence) - бізнес-аналітика; сукупність методів, інструментів та технологій для збору, інтеграції, аналізу та візуалізації бізнес-інформації з метою підтримки прийняття управлінських рішень.

DWH (Data Warehouse) - сховище даних; централізоване репозиторій структурованих даних, оптимізоване для аналітичної обробки та формування звітів.

ETL (Extract, Transform, Load) - процес вилучення даних з джерел, їх трансформації (очищення, нормалізації, агрегації) та завантаження у цільове сховище даних.

OLAP (Online Analytical Processing) - технологія багатовимірного аналізу даних, що дозволяє швидко виконувати складні запити та формувати звіти в режимі реального часу.

KPI (Key Performance Indicator) - ключовий показник ефективності; вимірювана величина, що демонструє ступінь досягнення визначених бізнес-цілей.

DAX (Data Analysis Expressions) - мова формул та функцій у Power BI, призначена для створення обчислюваних стовпців, мір та таблиць у моделях даних.

CSV (Comma-Separated Values) - текстовий формат представлення табличних даних, у якому значення полів розділяються комами або іншими роздільниками.

SQL (Structured Query Language) - стандартизована мова запитів для роботи з реляційними базами даних: створення, читання, оновлення та видалення даних.

API (Application Programming Interface) - програмний інтерфейс додатка; набір правил та протоколів для взаємодії між програмними компонентами.

ВСТУП

Актуальність теми дослідження. У сучасних умовах цифрової трансформації роздрібна торгівля генерує великі обсяги даних про транзакції, поведінку клієнтів, товарні запаси та фінансові показники. Ефективне використання цих даних є критичним фактором конкурентоспроможності бізнесу. Традиційні методи звітності не забезпечують необхідної швидкості та глибини аналізу для прийняття оперативних управлінських рішень. Бізнес-аналітика (Business Intelligence, BI) надає інструменти для візуалізації, агрегації та прогнозування ключових показників ефективності (KPI), що дозволяє виявляти тренди, оптимізувати асортимент та підвищувати рентабельність. Розробка спеціалізованого BI-додатку для роздрібної торгівлі є актуальним завданням, що поєднує сучасні технології обробки даних, машинного навчання та інтерактивної візуалізації.

Мета дослідження - розробити BI-додаток для комплексного аналізу та візуалізації показників продажів у роздрібній торгівлі, що забезпечує підтримку прийняття управлінських рішень на основі даних.

Завдання дослідження:

1. Проаналізувати предметну область роздрібної торгівлі та існуючі BI-рішення, визначити функціональні вимоги до системи.
2. Обґрунтувати вибір технологічного стеку: PostgreSQL для зберігання даних, Python для ETL-обробки та аналітики, Power BI для візуалізації.
3. Спроєктувати схему бази даних та реалізувати ETL-конвеєр для завантаження, очищення та валідації даних із CSV-джерел.
4. Розробити аналітичні модулі: прогнозування продажів (Prophet), ABC-класифікацію товарів, сегментацію клієнтів.
5. Створити інтерактивні дашборди в Power BI з DAX-метриками для моніторингу виручки, прибутку, динаміки продажів та поведінки клієнтів.
6. Провести тестування системи: модульне тестування ETL-трансформацій, валідацію синтаксису та логіки DAX-мір, перевірку коректності візуалізації.

Об'єкт дослідження - процес аналізу даних продажів у роздрібній торгівлі для підтримки управлінських рішень.

Предмет дослідження - методи, алгоритми та програмні засоби розробки BI-додатку для візуалізації та прогнозування комерційних показників.

Теоретична цінність роботи полягає в аналізі та порівнянні архітектурних підходів до побудови ETL-конвеєрів на базі Python з використанням бібліотек pandas та SQLAlchemy, дослідженні методів прогнозування часових рядів (Prophet) для задач роздрібної торгівлі, а також систематизації DAX-метрик у Power BI для розрахунку ключових показників ефективності [23].

Практичне значення одержаних результатів полягає у створенні програмного комплексу, що реалізує повний цикл аналізу даних: від завантаження та очищення даних до їх візуалізації у вигляді інтерактивних дашбордів. Розроблена система може бути використана як основа для подальших досліджень у галузі бізнес-аналітики, а також як навчальний матеріал при вивченні технологій PostgreSQL, Python та Power BI.

Структура кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних посилань та додатків. У першому розділі проведено аналіз предметної області та існуючих BI-рішень, сформульовано вимоги до системи. Другий розділ присвячено проєктуванню архітектури, моделюванню даних та розробці алгоритмів аналітики. Третій розділ описує програмну реалізацію, тестування та експлуатацію системи.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття та принципи Business Intelligence

Business Intelligence (BI) - це сукупність технологій, архітектурних рішень та бізнес-процесів, спрямованих на збір, інтеграцію, аналіз та візуалізацію даних для підтримки прийняття управлінських рішень. Основою функціонування будь-якої BI-системи є чотири взаємопов'язані компоненти: сховище даних, ETL-процеси, OLAP-аналітика та інтерактивна візуалізація.

Data Warehousing (Сховище даних) - централізоване репозиторій, оптимізоване для аналітичних запитів та формування звітів. На відміну від операційних баз даних (OLTP), орієнтованих на швидку транзакційну обробку, DWH зберігає історичні, очищені та структуровані дані. Найпоширенішою моделлю є схема «зірка» (star schema), що складається з центральної таблиці фактів та пов'язаних таблиць вимірів, що забезпечує швидку агрегацію та спрощує побудову звітів.

ETL-процеси (Extract, Transform, Load) - технологічний конвеєр підготовки даних. На етапі Extract дані вилучаються з гетерогенних джерел (ERP, CRM, CSV, API). Етап Transform включає очищення від помилок, нормалізацію форматів, валідацію типів, заповнення пропусків та обчислення похідних метрик. На етапі Load структуровані дані завантажуються у DWH із дотриманням посилальної цілісності та індексацією для оптимізації подальших запитів.

OLAP (Online Analytical Processing) - технологія багатовимірного аналізу, що дозволяє виконувати складні запити до даних у режимі, близькому до реального часу. Ключові операції OLAP включають: *slice* (фільтрація за одним виміром), *dice* (вибір підмножини за кількома вимірами), *drill-down* (деталізація до нижчого рівня ієрархії) та *roll-up* (агрегація до вищого рівня). У сучасних BI-платформах OLAP реалізовано через in-memory движки, що забезпечують миттєвий перерахунок показників при зміні фільтрів.

KPI (Key Performance Indicators) - ключові показники ефективності, що кількісно відображають ступінь досягнення стратегічних цілей. У роздрібній торгівлі до базових KPI належать: загальна виручка, рентабельність продажів, середній чек, оборотність запасів, конверсія та рівень утримання клієнтів. KPI слугують стандартизованими метриками для оцінки результатів діяльності та виявлення зон для оптимізації.

Дашборди - інтерактивні панелі моніторингу, що агрегують KPI, графіки, таблиці та фільтри на одному екрані. Ефективний дашборд відповідає принципам візуальної лаконічності, контекстної релевантності та підтримки навігації (cross-filtering, drill-through), дозволяючи користувачам оперативно відстежувати тренди, виявляти аномалії та прийняти обґрунтовані управлінські рішення.

Таким чином, сучасна BI-архітектура базується на послідовній трансформації сирих даних у структуроване сховище, їх багатовимірному аналізі та поданні результатів у зрозумілій візуальній формі. Дотримання цих принципів забезпечує створення відмово- та масштабованої аналітичної системи, адаптованої до специфіки роздрібної торгівлі.

1.2. Огляд існуючих BI-рішень

Сучасний ринок інструментів бізнес-аналітики пропонує широкий спектр рішень, що відрізняються за функціональністю, вартістю, складністю впровадження та цільовою аудиторією. Для обґрунтування вибору технологічного стеку у даній роботі проведено порівняльний аналіз чотирьох популярних платформ: Microsoft Power BI, Tableau, Metabase та Apache Superset. Порівняльна характеристика основних критеріїв наведена в таблиці 1.1.

Розгляд придатності сучасних BI-платформ для малого та середнього бізнесу виявляє різні профілі ефективності. Microsoft Power BI вирізняється низькою стартовою вартістю та інтуїтивним інтерфейсом, а також глибокою інтеграцією з екосистемою Microsoft, що забезпечує регулярні оновлення та широку навчальну базу. Проте його застосування може ускладнюватися через високий поріг входження до мови DAX для нетехнічних фахівців, обмеження

безкоштовної версії щодо публікації звітів у хмарі та залежність від корпоративного середовища.

Таблиця 1.1 – Порівняльна характеристика BI-платформ

Критерій	Power BI	Tableau	Metabase	Apache Superset
Тип ліцензії	Пропрієтарна (Free/Pro/Premium)	Пропрієтарна (Creator/Explorer/Viewer)	Open Source (AGPL) + Cloud	Open Source (Apache 2.0)
Вартість для SMB	Низька (від $10/корист./міс)	Висока (від $70/корист./міс)	Безкоштовно (самохостинг)	Безкоштовно (самохостинг)
Підтримка української мови	Повна (інтерфейс, локалізація)	Часткова	Через переклади спільноти	Через переклади спільноти
Інтеграція з PostgreSQL	Нативна, стабільна	Нативна, потужна	Нативна, проста	Нативна, гнучка
Мова аналітики	DAX (потужна, складна)	LOD Expressions (гнучка)	Прості фільтри + SQL	SQL Lab + візуальний конструктор
Поріг входження	Середній	Високий	Низький	Середній/Високий
Можливості візуалізації	Широкий набір, кастомні візуали	Найбагатші, висока гнучкість	Базові, зрозумілі	Широкий набір, підтримка Plotly
Спільна робота	Інтеграція з Microsoft 365	Tableau Server/Online	Базова (коментарі, підписки)	Ролевий доступ, RSS
Мобільний додаток	Нативний (iOS/Android)	Нативний (iOS/Android)	Адаптивний веб-інтерфейс	Адаптивний веб-інтерфейс
Підтримка спільноти	Велика, офіційна документація	Велика, активна спільнота	Зростаюча, open-source	Активна, Apache Foundation
Масштабованість	Azure-інтеграція, Premium-ємності	Tableau Server, кластеризація	Обмежена для великих навантажень	Горизонтальне масштабування

Джерело: розроблено автором

Tableau пропонує найпотужніші інструменти візуалізації та високу гнучкість у створенні складних дашбордів із відмінною продуктивністю на великих масивах даних, однак висока вартість ліцензій, значна складність освоєння та необхідність виділеного сервера для командної роботи роблять його менш доступним для SMB-сектору. Metabase, зі свого боку, є привабливим завдяки безкоштовній self-hosted версії, мінімальному порогу входження за рахунок інтерфейсу типу «запитай і отримай» та простій інтеграції з популярними СУБД, проте обмежується базовими можливостями кастомізації візуалізацій, слабшою підтримкою складних обчислень та менш розвиненою спільнотою порівняно з ринковими лідерами. Apache Superset повністю безкоштовний, підтримує широкий спектр джерел даних, відрізняється високою гнучкістю конфігурації та активною розробкою, але вимагає значних технічних навичок для розгортання та супроводу, має недостатньо структуровану документацію для початківців та менш інтуїтивний користувацький інтерфейс.

Для реалізації BI-додатку в межах кваліфікаційної роботи обрано платформу Microsoft Power BI, що зумовлено комплексом технічних та експлуатаційних переваг [6]. Насамперед, інструмент забезпечує оптимальний баланс функціональності та доступності, пропонуючи потужні аналітичні можливості, зокрема підтримку мови DAX, інтеграцію з Python/R та роботу з часовими рядами, при помірній вартості, що є критичним для навчальних цілей. Важливим аргументом є нативна сумісність з обраним технологічним стеком: платформа безпосередньо підтримує PostgreSQL, дозволяє імпорт даних із CSV/Excel та забезпечує взаємодію з Python-скриптами, використаними для ETL-процесів та прогнозування. Додатково, повна українська локалізація інтерфейсу спрощує презентацію результатів в україномовному освітньому середовищі. Впровадження цифрових технологій в бізнес-процеси відповідає стратегічним напрямам Концепції розвитку цифрової економіки України [3]. Наявність безкоштовної версії Power BI Desktop дозволяє повноцінно розробляти, тестувати та налагоджувати дашборди без фінансових витрат, а широка офіційна

документація та активна спільнота користувачів суттєво прискорюють процес освоєння інструменту та вирішення технічних завдань.

Проведений аналіз ринку BI-рішень засвідчує, що сучасний інструментарій охоплює рішення різного рівня: від корпоративних enterprise-платформ, таких як Tableau та Power BI Premium, до відкритих open-source альтернатив на кшталт Metabase та Apache Superset. Для малого та середнього бізнесу пріоритетними критеріями вибору залишаються сукупна вартість володіння, простота інтеграції в існуючу IT-інфраструктуру та достатність базових аналітичних сценаріїв для оперативного моніторингу. У контексті даної роботи Microsoft Power BI демонструє оптимальне поєднання функціональної насиченості, економічної доступності та технічної сумісності з реалізованим стеком на базі PostgreSQL, Python та DAX. Обрана платформа дозволяє повноцінно реалізувати всі заплановані функціональні вимоги системи, зокрема автоматизовану інтеграцію даних, статистичне прогнозування, ABC-аналіз, сегментацію клієнтів та інтерактивну візуалізацію ключових показників ефективності.

1.3. Специфіка даних роздрібної торгівлі

Дані роздрібної торгівлі характеризуються високою частотою оновлення, значним обсягом транзакцій та чіткою реляційною структурою, яка зазвичай реалізується за моделлю «зірка». Центральним елементом такої архітектури виступає таблиця фактів продажів, що містить ідентифікатори товарів, клієнтів, торговельних точок, дати операцій, кількісні та фінансові показники. До неї приєднуються таблиці вимірів: довідники номенклатури товарів із категоризацією та ціноутворенням, профілі покупців з демографічними характеристиками та реєстри магазинів із географічною прив'язкою. Подібна організація забезпечує цілісність даних, спрощує агрегацію показників на різних рівнях ієрархії та оптимізує виконання складних аналітичних запитів.

Оцінка ефективності роздрібної діяльності базується на системі взаємопов'язаних метрик, що відображають фінансові результати та поведінкові патерни споживачів. Базовими показниками виступають загальний оборот,

валова маржа, середній чек та рівень повторних покупок, які дозволяють відстежувати динаміку виручки та рентабельність операцій у розрізі періодів або каналів продажу. Важливим інструментом стратегічного управління є розрахунок життєвої цінності клієнта (LTV), що визначає сукупний дохід від покупця за весь період взаємодії. Аналіз цих метрик у комбінації з часовими рядами формує інформаційну основу для оперативного коригування цінової політики та асортиментної стратегії [4].

Окрім базових фінансових показників, роздрібна аналітика активно використовує методи класифікації та сегментації для оптимізації ресурсів та персоналізації маркетингу. Серед найбільш поширених підходів виділяють:

– ABC-аналіз, заснований на принципі Парето, який розподіляє товари на три категорії за внеском у виручку (близько 80 %, 15 % та 5 % відповідно) та дозволяє зосередити управлінську увагу на найприбутковіших позиціях [22];

– Сегментацію клієнтів за рівнем витрат та частотою покупок, що формує ієрархію від стандартних до віп-покупців і забезпечує таргетоване комунікаційне планування;

– Ковзне усереднення та порівняння періодів (MoM, YoY), які згладжують короткострокові флуктуації та виділяють стійкі тренди попиту.

Специфіка роздрібних даних також визначається наявністю виражених сезонних коливань, впливом зовнішніх факторів та високою ймовірністю появи аномалій або пропусків у первинних записах. Це вимагає обов'язкового етапу очищення та валідації на рівні ETL-конвеєра, включно з перевіркою посилальної цілісності, нормалізацією текстових полів, обробкою некоректних дат та автоматичним коригуванням цінових розбіжностей. Крім того, для точного планування закупівель та логістики необхідне застосування статистичних моделей прогнозування часових рядів, які враховують тренди, сезонність та календарні ефекти. У підсумку, побудова ефективної BI-системи для роздрібної торгівлі вимагає поєднання надійного сховища даних, автоматизованих пайплайнів трансформації та інструментів інтерактивної візуалізації, адаптованих до специфіки комерційної аналітики.

1.4. Висновки до розділу та формулювання технічних вимог

Проведений теоретичний та аналітичний огляд предметної області дозволив сформулювати ключові положення щодо побудови BI-системи для роздрібної торгівлі. По-перше, визначено, що ефективна бізнес-аналітика базується на чіткій архітектурі, яка поєднує автоматизований ETL-конвеєр, реляційне сховище даних, механізми багатовимірного аналізу (OLAP) та інтерактивну візуалізацію. По-друге, порівняльний аналіз сучасних інструментів (Power BI, Tableau, Metabase, Apache Superset) продемонстрував, що для малого та середнього бізнесу оптимальним є баланс між функціональністю, вартістю володіння та простотою інтеграції. На цій основі обрано Microsoft Power BI як платформу візуалізації, а Python разом із PostgreSQL – як стек для обробки та зберігання даних. По-третє, дослідження специфіки роздрібних даних підтвердило необхідність використання моделі «зірка», реалізації ключових комерційних метрик (оборот, маржа, LTV, конверсія) та впровадження спеціалізованих аналітичних модулів (ABC-класифікація, сегментація клієнтів, прогнозування часових рядів).

На основі проведеного аналізу сформовано перелік функціональних та нефункціональних вимог до розроблюваної BI-системи.

Функціональні вимоги:

1. Реалізація ETL-конвеєра для автоматизованого вилучення, очищення, валідації та завантаження даних із файл-джерел (CSV/Excel) у цільову базу даних.
2. Побудова реляційної моделі даних за схемою «зірка» з центральною таблицею фактів (sales) та таблицями вимірів (products, customers, stores, date).
3. Інтеграція аналітичних модулів: прогнозування попиту на 12 місяців вперед, ABC-класифікація товарів за принципом Парето, сегментація клієнтів за рівнем витрат та частотою покупок.
4. Створення інтерактивних дашбордів із візуалізацією KPI: загальна виручка, прибуток, рентабельність, середній чек, динаміка MoM/YoY, накопичувальний підсумок.

5. Підтримка багатовимірної фільтрації та агрегації даних за часовими інтервалами, регіонами, категоріями товарів, магазинами та рівнями клієнтів.

6. Можливість експорту аналітичних звітів та результатів класифікації у стандартні формати (CSV, PNG, PDF).

Нефункціональні вимоги:

1. Продуктивність: час відгуку інтерфейсу при зміні фільтрів на дашборді не повинен перевищувати 2 секунди; час виконання повного ETL-циклу для набору даних обсягом до 50 000 записів – не більше 5 хвилин.

2. Надійність: забезпечення цілісності даних через транзакційне завантаження, перевірку зовнішніх ключів, обробку аномалій та детальне логування етапів трансформації.

3. Масштабованість: архітектура повинна підтримувати інкрементальне оновлення даних, індексацію за критичними полями та можливість горизонтального розширення сховища при зростанні обсягів транзакцій.

4. Безпека: реалізація контрольованого доступу до даних, захист персональної інформації клієнтів відповідно до Закону України «Про захист персональних даних» [2] шляхом валідації та обмеження прямого експорту чутливих полів.

5. Ергономіка: інтуїтивний інтерфейс візуалізації, відповідність сучасним стандартам UX для аналітичних панелей, повна підтримка української мови в інтерфейсі та документації.

Сформульовані вимоги слугують технічним базисом для проєктування архітектури системи, розробки моделей даних, алгоритмів аналітики та інтеграції програмних компонентів. Їхня практична реалізація, обґрунтування технологічних рішень та перевірка коректності роботи системи будуть детально розглянуті в наступному розділі кваліфікаційної роботи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА АЛГОРИТМІВ

2.1. Вибір технологічного стеку та обґрунтування архітектурних рішень

Проєктування архітектури BI-системи базується на принципах модульності, масштабованості та чіткого розділення відповідальності між компонентами збору, трансформації, зберігання та візуалізації даних. Загальна схема потоку даних реалізована за послідовною моделлю: джерела сирих даних (CSV/Excel) → автоматизований ETL-конвеєр на мові Python → реляційне сховище даних PostgreSQL → аналітична модель та інтерактивні дашборди в Power BI. Загальна архітектурна схема взаємодії компонентів наведена на рисунку 2.1. Така побудова забезпечує ізоляцію етапів підготовки даних від етапів їхнього візуального подання, що спрощує підтримку системи, дозволяє незалежно масштабувати окремі вузли та мінімізує ризики втрати цілісності інформації.

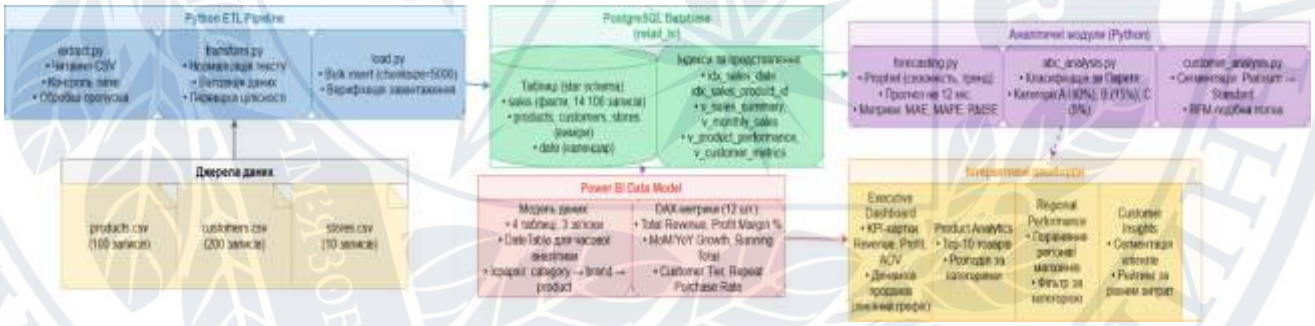


Рисунок 2.1 – Архітектурна схема BI-додатку

Для зберігання та управління структурованими даними обрано реляційну систему керування базами даних PostgreSQL. Вибір зумовлений повною підтримкою стандартів ACID, розвиненою системою індексації, гнучкими механізмами забезпечення посилальної цілісності (обмеження FOREIGN KEY з правилами ON DELETE RESTRICT / ON UPDATE CASCADE) та наявністю вбудованих функцій для роботи з часовими рядами та агрегації. У контексті

роздрібної аналітики PostgreSQL ефективно обслуговує модель «зірка», де центральна таблиця фактів sales пов'язана з довідниками products, customers та stores. Створені композитні індекси за полями sale_date та ідентифікаторами сутностей забезпечують оптимальну швидкість виконання аналітичних запитів, що є критичним для інтерактивної роботи BI-платформи та обробки понад 397 тисяч транзакційних записів.

Реалізація конвеєра підготовки даних та аналітичних модулів виконана з використанням екосистеми Python. Бібліотека pandas [16] забезпечує ефективну векторизовану обробку табличних даних, включаючи нормалізацію текстових полів, валідацію типів, обробку пропущених значень та фільтрацію дублікатів [9]. Для взаємодії з базою даних використано SQLAlchemy у поєднанні з драйвером psycopg2, що дозволяє реалізувати механізм пакетного завантаження (chunksize = 5000) з контрольованою транзакційністю, що суттєво підвищує продуктивність на великих обсягах даних [18]. Аналітична складова базується на бібліотеці Prophet для прогнозування тимчасових рядів з урахуванням адитивної/мультиплікативної сезонності та трендів, а також matplotlib для генерації візуалізацій ABC-класифікації та сегментації клієнтів [8].Вихідний код моделі Prophet доступний у відкритому репозиторії [11]. Використання відкритого програмного забезпечення дозволяє інтегрувати статистичні моделі без додаткових ліцензійних витрат та забезпечує відтворюваність результатів на різних середовищах.

Платформою для фінального представлення даних обрано Microsoft Power BI Desktop [12]. Обґрунтування цього рішення базується на широких можливостях імпорту даних із зовнішніх реляційних джерел, наявності потужної мови обчислень DAX (Data Analysis Expressions) для реалізації складних бізнес-логік та інтерактивних механізмів крос-фільтрації. Мова DAX дозволяє динамічно обчислювати метрики в контексті обраних користувачем фільтрів (наприклад, MoM/YoY зростання, накопичувальні підсумки, ранжування магазинів), що є неможливим при використанні статичних SQL-представлень. Інтеграція з календарною таблицею DateTable забезпечує коректну роботу

функцій часового зсуву (PREVIOUSMONTH, SAMEPERIODLASTYEAR, DATESYTD), що критично для фінансової та операційної аналітики роздрібної торгівлі.

Інтеграція обраних технологій реалізована через чітко визначені інтерфейси обміну даними: ETL-конвеєр експортує очищені набори у реляційне сховище, Power BI підключається до нього в режимі імпорту для побудови семантичної моделі, а результати аналітичних обчислень (прогнози, класифікації) завантажуються як додаткові таблиці. Такий підхід гарантує узгодженість метрик між різними компонентами системи, унеможливлює дублювання бізнес-логіки та забезпечує єдине джерело істини для всіх звітів. Архітектура відповідає нефункціональним вимогам щодо надійності, продуктивності та зручності експлуатації, що підтверджується результатами модульного та інтеграційного тестування, описаними в розділі 3.

Таким чином, обраний технологічний стек поєднує надійність реляційного сховища даних, гнучкість програмного середовища для автоматизованої підготовки та статистичного аналізу, а також інтерактивні можливості сучасної BI-платформи. Впроваджена архітектура забезпечує масштабованість системи, мінімізує ручні операції з обробки даних та створює технічну основу для реалізації всіх функціональних вимог, сформульованих у першому розділі роботи.

2.2. Моделювання даних та проєктування бази даних

Проєктування бази даних для BI-системи роздрібної торгівлі базується на принципах багатовимірного моделювання, що забезпечує ефективну агрегацію даних та швидке виконання аналітичних запитів. Концептуальна модель даних визначає чотири основні сутності: products (товари), customers (клієнти), stores (магазини) та sales (транзакції), зв'язки між якими відображено на рисунку 2.2. Центральне місце у моделі займає таблиця фактів sales, що пов'язана зовнішніми ключами з таблицями вимірів, формуючи класичну схему «зірка» (star schema) [7]. Така структура спрощує побудову ієрархій (наприклад, category → brand →

product), забезпечує денормалізацію вимірів для прискорення запитів та підтримує цілісність даних через механізми посилальної валідації.

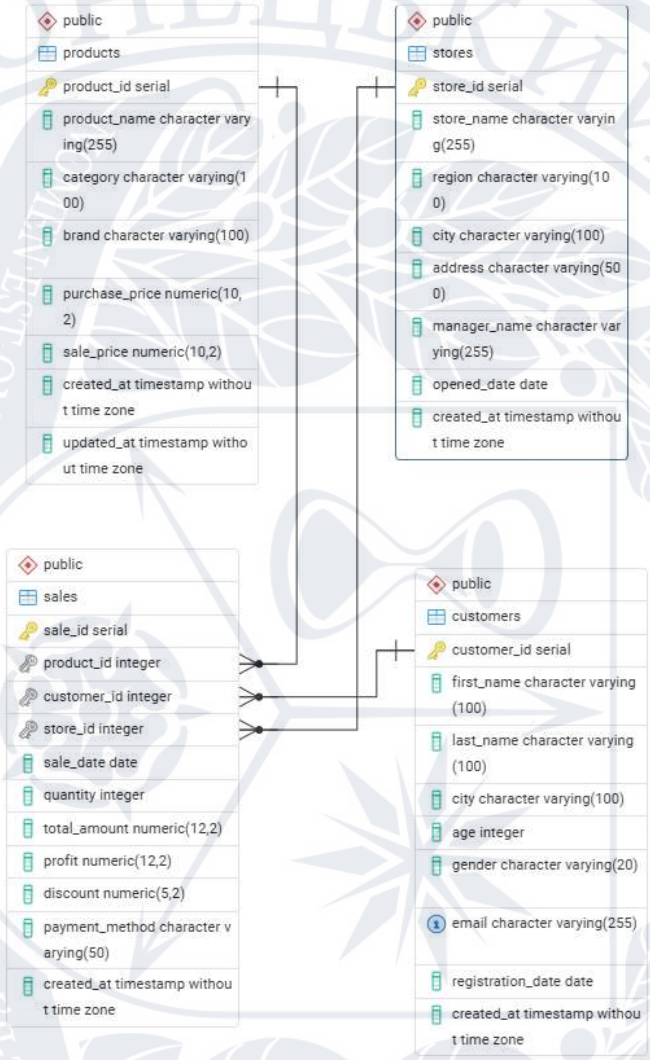


Рисунок 2.2 – ER-діаграма бази даних роздрібної торгівлі

Логічна модель реалізована у реляційній СУБД PostgreSQL з чітким розмежуванням типів даних та обмежень цілісності [17]. Таблиця фактів sales містить ідентифікатори зовнішніх сутностей (product_id, customer_id, store_id), дату транзакції, кількісні та фінансові показники (quantity, total_amount, profit, discount). Для забезпечення коректності даних застосовано обмеження CHECK: quantity > 0, total_amount >= 0, discount BETWEEN 0 AND 100. Зовнішні ключі налаштовано з правилами ON DELETE RESTRICT та ON UPDATE CASCADE,

що запобігає видаленню довідкових записів, на які посилаються транзакції, та автоматично оновлює посилання при зміні первинних ключів. Таблиці вимірів містять атрибути, необхідні для фільтрації та групування: категорії товарів, демографічні дані клієнтів, географічна прив'язка магазинів.

Фізична реалізація бази даних оптимізована для аналітичних навантажень шляхом створення стратегічних індексів. Для таблиці sales визначено індекси за полями sale_date, product_id, customer_id, store_id, а також композитні індекси (sale_date, product_id) та (sale_date, store_id) для прискорення запитів із фільтрацією за часом та групуванням за товаром або магазином. Таблиці вимірів індексовані за полями, що часто використовуються у WHERE та JOIN: category, brand, city, region. Додатково реалізовано тригер update_updated_at_column() для автоматичного оновлення метаданих зміни записів, що спрощує аудит та синхронізацію даних.

Для спрощення формування звітів та візуалізації у Power BI створено чотири денормалізовані представлення (views). Представлення v_sales_summary об'єднує дані з усіх таблиць в єдиний набір із розрахованими полями (рік, місяць, квартал, повне ім'я клієнта), що усуває необхідність складних JOIN у запитах дашбордів. v_monthly_sales агрегує продажі за місяцями, формуючи базовий набір метрик для прогнозування: загальна виручка, прибуток, кількість транзакцій, середній чек. v_product_performance розраховує KPI по товарах (загальна виручка, прибуток, маржа), що є основою для ABC-аналізу. v_customer_metrics формує профілі клієнтів із розрахунком сумарних витрат, частоти покупок та автоматичною сегментацією за рівнем лояльності. Використання представлень дозволяє інкапсулювати складну бізнес-логіку на рівні бази даних, забезпечуючи узгодженість метрик між різними компонентами системи та спрощуючи підтримку коду.

Таким чином, розроблена модель даних поєднує теоретичні принципи багатовимірного проєктування з практичними вимогами до продуктивності та цілісності. Схема «зірка» з оптимізованими індексами та денормалізованими представленнями забезпечує швидкий доступ до агрегованих показників, що є

критичним для інтерактивної роботи BI-платформи. Реалізовані обмеження та тригери гарантують коректність даних на рівні СУБД, мінімізуючи ризики помилок у бізнес-логіці. Ця архітектура слугує надійним фундаментом для реалізації аналітичних модулів та візуалізації, описаних у наступних підрозділах.

2.3. Проєктування ETL-конвеєра

ETL-конвеєр (Extract, Transform, Load) є критичним компонентом BI-архітектури, що забезпечує автоматизовану підготовку сирих даних до аналітичної обробки. У розробленій системі реалізовано модульний ETL-пайплайн мовою Python, який відповідає принципам єдиної відповідальності, конфігурованості та відтворюваності результатів. Загальна схема взаємодії модулів конвеєра наведена на рисунку 2.3. Проєктування базується на чіткому розділенні трьох основних фаз, кожна з яких інкапсульована у окремий клас із визначеним інтерфейсом, що забезпечує незалежність етапів, спрощує підтримку коду та дозволяє масштабувати окремі компоненти без впливу на загальну архітектуру системи.

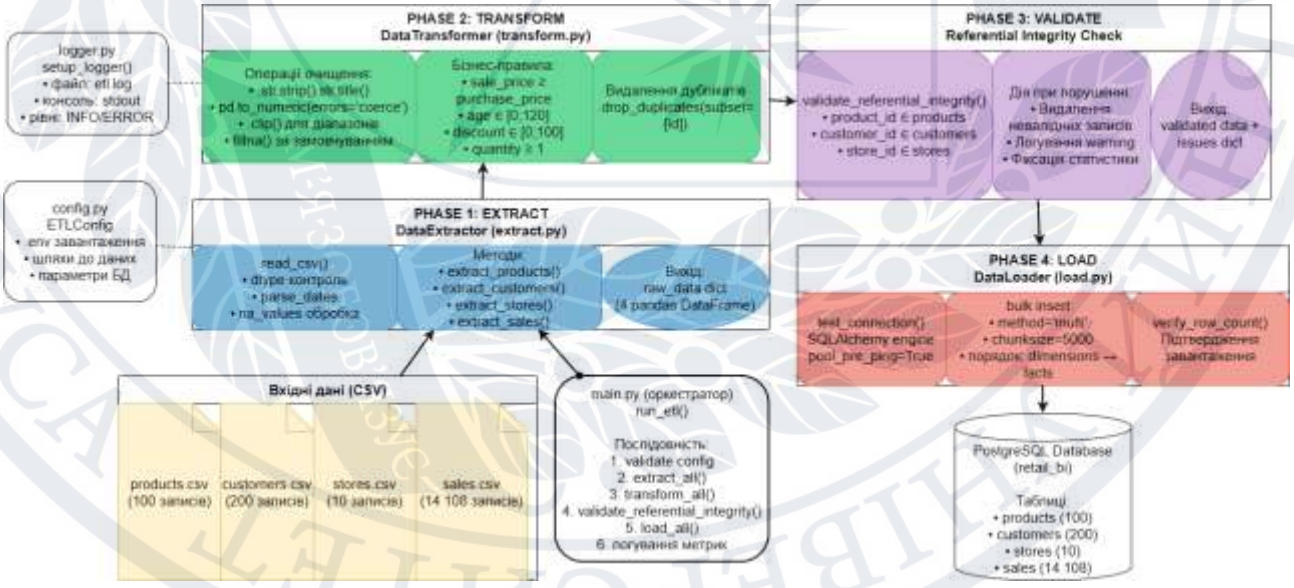


Рисунок 2.3 – Схема взаємодії модулів ETL-конвеєра

Фаза Extract реалізована у класі DataExtractor (модуль etl/extract.py) і виконує зчитування даних із чотирьох CSV-джерел із примусовим контролем

типів (dtype для int64, parse_dates для sale_date) та уніфікованою інтерпретацією відсутніх значень через параметр na_values, що гарантує коректний парсинг незалежно від формату запису пропусків. Фаза Transform у класі DataTransformer охоплює нормалізацію текстових полів (.str.strip().str.title()), валідацію числових діапазонів (pd.to_numeric(errors="coerce").clip()), застосування бізнес-правил (коригування цін, заповнення пропусків за замовчуванням) та фільтрацію дублікатів за первинними ключами. Фаза Load у класі DataLoader використовує ORM SQLAlchemy для оптимізованої пакетної вставки (method="multi", chunksize=5000) з обов'язковим дотриманням послідовності завантаження: спочатку таблиці вимірів (products, customers, stores), потім таблиця фактів sales, що гарантує цілісність посилань на рівні СУБД.

Процес трансформації реалізовано як детермінований алгоритм із чіткою послідовністю кроків, що забезпечує стабільність результатів незалежно від стану вхідних даних. Для довідників products, customers та stores застосовано стандартизацію текстових атрибутів, валідацію цінових та демографічних діапазонів із автоматичним виправленням логічних помилок (наприклад, sale_price < purchase_price коригується множенням на 1.15). Особливу увагу приділено таблиці sales, де алгоритм включає відкидання записів із некоректними датами (pd.to_datetime(errors="coerce") + dropna), валідацію кількості та сум із обмеженням мінімальних значень, нормалізацію способів оплати, хронологічне сортування та видалення повторюваних транзакцій, що є критичним для подальшої побудови точних часових рядів у Power BI.

Для мінімізації ризиків потрапляння некоректних даних у цільове сховище реалізовано багаторівневу систему контролю якості. Перевірка посилальної цілісності методом validate_referential_integrity() порівнює зовнішні ключі таблиці фактів із множинами допустимих ідентифікаторів вимірів, автоматично відфільтровуючи невалідні посилання з логуванням попереджень. Процес супроводжується двоканальним логуванням через модуль logger.py (консоль для оперативного моніторингу, файл etl.log для аудиту) з фіксацією часових міток, рівнів подій (INFO, WARNING, ERROR) та контексту модулів. Обробка

відхилень побудована на принципі "fail-fast": критичні помилки (відсутність файлів, розрив з'єднання з БД) призводять до зупинки конвеєра, тоді як некритичні попередження фіксуються для подальшого аналізу без переривання виконання.

Оркестрація виконання реалізована в модулі main.py через функцію run_etl(), яка послідовно координує валідацію конфігурації, фази Extract, Transform, Validate та Load із фіксацією метрик продуктивності (час, кількість оброблених записів, статистика помилок). Параметри підключення, шляхи до даних та розмір пакету винесено у клас ETLConfig і завантажуються з файлу .env через python-dotenv, що відповідає архітектурному принципу "12-factor app" та дозволяє гнучко адаптувати систему під різні середовища без модифікації коду [20]. Така структура забезпечує повну прозорість процесу, підтримку поетапного тестування окремих модулів та високу стійкість до змін у вхідних даних, створюючи надійний технічний фундамент для подальшої аналітичної обробки.

2.4. Алгоритми аналітичної обробки

Алгоритми аналітичної обробки є ключовим компонентом BI-системи, що перетворює структуровані дані у змістовні бізнес-інсайти. У розробленій системі реалізовано чотири основні аналітичні модулі: ABC-класифікацію товарів, сегментацію клієнтів, прогнозування часових рядів та розрахунок динамічних метрик через DAX. Кожен алгоритм спроектовано з урахуванням специфіки роздрібної торгівлі, вимог до інтерпретованості результатів та інтеграції з інтерфейсом Power BI.

Алгоритм ABC-класифікації товарів базується на принципі Парето та реалізується у модулі abc_analysis.py. Після агрегації виручки за кожним товаром (SUM(total_amount) GROUP BY product_id) виконується сортування за спаданням та розрахунок накопиченої частки: cumulative_pct = RUNNING_SUM(revenue) / TOTAL(revenue) * 100. Класифікація визначається пороговими значеннями: категорія A (товари, що формують перші 80% виручки), категорія B (наступні 15%, до 95% сукупної виручки), категорія C (решта 5%).

Такий підхід дозволяє автоматично виділяти пріоритетні товарні позиції для оптимізації запасів, ціноутворення та маркетингових активностей, а результати візуалізуються у вигляді кривої Парето та експортуються у CSV для подальшого використання.

Сегментація клієнтів реалізована у модулі customer_analysis.py на основі двовимірної матриці «сукупні витрати × частота покупок». Рівень клієнта (tier) визначається за пороговими значеннями загальної виручки: Platinum (≥5000), Gold (≥3000), Silver (≥1500), Bronze (≥500), Standard (<500). Паралельно розраховується тип залученості (engagement) за кількістю покупок: VIP (≥10), Frequent (6–9), Regular (3–5), Occasional (1–2), At Risk (0). Алгоритм використовує функції pd.cut() для бінінгу та groupby().agg() для агрегації метрик, що забезпечує масштабованість на великих вибірках [14]. Результати сегментації інтегруються у дашборд Customer Insights для таргетованого маркетингу та аналізу лояльності.

Прогнозування продажів виконується за допомогою бібліотеки Prophet (Meta) у модулі forecasting.py [10]. Модель базується на адитивній декомпозиції часового ряду: $y(t) = trend(t) + seasonality(t) + holidays(t) + \varepsilon$. Для врахування річної та місячної сезонності задано параметри yearly_seasonality=True, seasonality_mode='multiplicative', а також додано кастомну сезонність з періодом 30.5 днів. Горизонт прогнозу становить 12 місяців, довірчий інтервал - 95% (interval_width=0.95), параметр changepoint_prior_scale=0.05 регулює гнучкість тренду. Точність моделі оцінюється метриками MAE, MAPE, RMSE, що розраховуються на історичних даних шляхом порівняння фактичних та прогнозних значень [21]. Результати експортуються у вигляді графіків та таблиць для інтеграції у Power BI.

Логіка розрахунку динамічних метрик реалізована через мову DAX у Power BI та забезпечує інтерактивне обчислення показників у контексті обраних фільтрів. Ключові алгоритми включають: розрахунок MoM-зростання через функцію PREVIOUSMONTH, YoY-порівняння через SAMEPERIODLASTYEAR, накопичувального підсумку через DATESYTD, а також ковзного середнього через AVERAGEX у поєднанні з DATESINPERIOD. Для уникнення помилок

ділення на нуль застосовано функцію DIVIDE(чисельник, знаменник, 0). Усі міри побудовано з урахуванням контексту фільтрації, що дозволяє користувачам динамічно змінювати період, регіон чи категорію без необхідності перерахунку на стороні сервера.

Таким чином, реалізовані алгоритми аналітичної обробки поєднують статистичну обґрунтованість, бізнес-інтерпретованість та технічну ефективність. ABC-класифікація та сегментація клієнтів забезпечують стратегічне планування асортименту та маркетингу, модель Prophet дозволяє прогнозувати попит з оцінкою невизначеності, а DAX-метрики надають інструменти для оперативного моніторингу та порівняльного аналізу. Інтеграція цих алгоритмів у єдину архітектуру BI-системи створює комплексний інструмент підтримки управлінських рішень у роздрібній торгівлі.

2.5. Проєктування інтерфейсу та візуалізації

Проєктування інтерфейсу BI-додатку базується на принципах інформаційної архітектури, візуальної ієрархії та користувальницької ергономіки, що забезпечують ефективне сприйняття аналітичних даних та швидке прийняття управлінських рішень. У розробленій системі реалізовано чотири тематичні дашборди, кожен з яких орієнтований на конкретну аудиторію та сценарій використання: Executive Dashboard для моніторингу загальних фінансових показників, Product Analytics для аналізу асортименту та прибутковості товарів, Regional Performance для порівняльної оцінки ефективності торговельних точок та Customer Insights для сегментації клієнтів та аналізу лояльності. Така модульна структура дозволяє користувачам фокусуватися на релевантних метриках без інформаційного перевантаження, забезпечуючи послідовну навігацію від стратегічного огляду до операційної деталізації.

Вибір типів візуалізацій здійснювався відповідно до семантики даних та аналітичних завдань. Для відображення динаміки часових рядів використано лінійні графіки з можливістю порівняння фактичних та прогнозних значень, що

підтримується інтеграцією з модулем Prophet. Структурні співвідношення (частка категорій у виручці, розподіл клієнтів за рівнями) візуалізуються через кругові та кільцеві діаграми з чіткою легендою та підписами відсотків. Порівняльний аналіз реалізовано за допомогою стовпчикових гістограм із можливістю сортування та фільтрації, а деталізовані дані представлені у вигляді інтерактивних таблиць із підтримкою пагінації та експорту. Ключові показники ефективності (Total Revenue, Profit Margin %, Average Order Value) винесено у вигляді KPI-карток із кольоровою індикацією відхилень від цільових значень, що забезпечує миттєву оцінку стану бізнесу.

Механізми інтерактивності реалізовано засобами Power BI через крос-фільтрацію, drill-through навігацію та динамічні зрізи (slicers). Крос-фільтрація дозволяє виділити елемент на одному візуальному об'єкті (наприклад, категорію товару) та автоматично оновити всі пов'язані графіки, забезпечуючи контекстний аналіз без додаткових запитів. Функція drill-through надає можливість перейти від агрегованого показника (виручка за регіоном) до детального перегляду транзакцій конкретного магазину, зберігаючи контекст фільтрів. Динамічні зрізи реалізовано для ключових вимірів: часовий період (місяць/квартал/рік), географічна прив'язка (регіон/місто/магазин), товарна категорія та рівень клієнта, що дозволяє користувачам гнучко адаптувати вигляд дашборду під поточне аналітичне завдання.

Вимоги до адаптивності та експорту враховано на етапі проєктування макетів: всі дашборди розроблено з підтримкою респонсивної верстки, що забезпечує коректне відображення на екранах різної роздільної здатності (від моніторів до планшетів). Кольорова схема обрана з урахуванням доступності (contrast ratio $\geq 4.5:1$) та семантики (зелений - позитивна динаміка, червоний - відхилення, сірий - нейтральні значення), що сприяє інтуїтивному сприйняттю навіть користувачами з обмеженим досвідом роботи з аналітичними системами. Підтримка української локалізації реалізовано на рівні інтерфейсу: підписи осей, легенди, спливаючі підказки та кнопки експорту адаптовано державною мовою, що відповідає вимогам україномовного освітнього та бізнес-середовища.

Можливості експорту результатів аналізу інтегровано в кожен дашборд через стандартні функції Power BI: експорт візуалізацій у форматах PNG/PDF для включення у звіти та презентації, експорт таблиць у CSV для подальшої обробки в зовнішніх інструментах, а також генерація посилань для спільного доступу до інтерактивних дашбордів. Для забезпечення цілісності даних при експорті реалізовано механізм «заморожування» контексту фільтрів: експортований звіт містить не лише візуалізацію, але й метадані про застосовані фільтри, що гарантує відтворюваність результатів. Такий підхід поєднує інтерактивність онлайн-аналітики з можливістю офлайн-роботи та документування прийнятих рішень.

Таким чином, проєктування інтерфейсу та візуалізації в розробленій BI-системі реалізовано з урахуванням сучасних стандартів UX для аналітичних панелей, принципів доступності та вимог україномовної локалізації. Модульна структура дашбордів, семантично обґрунтований вибір типів графіків, гнучкі механізми інтерактивності та підтримка експорту створюють комплексне середовище для моніторингу, аналізу та документування комерційних показників роздрібної торгівлі, що відповідає функціональним вимогам, сформульованим у першому розділі роботи.

Висновки до розділу 2

Проведене проєктування архітектури та алгоритмів BI-додатку дозволило сформувати технічно обґрунтовану, масштабовану та цілісну основу для програмної реалізації системи. За результатами другого розділу сформульовано такі ключові положення:

1. Обраний технологічний стек (PostgreSQL, Python, Power BI) забезпечує оптимальний баланс між функціональністю, вартістю володіння та інтеграційною сумісністю. Нативна підтримка реляційного сховища, гнучкість екосистеми pandas/SQLAlchemy для підготовки даних та потужні аналітичні можливості DAX дозволяють реалізувати повний цикл BI-системи без надмірних ліцензійних витрат.

2. Реляційна модель даних за схемою «зірка» з центральною таблицею фактів sales та таблицями вимірів products, customers, stores, date оптимізована для аналітичних навантажень. Стратегічне індексування, обмеження цілісності та денормалізовані представлення (v_sales_summary, v_monthly_sales тощо) забезпечують швидкий доступ до агрегованих метрик та мінімізують складність JOIN-операцій у запитах.

3. Модульна архітектура ETL-конвеєра з чітким розділенням фаз Extract, Transform, Load та Validate гарантує високу якість підготовлених даних. Імплементація нормалізації текстових полів, валідації числових діапазонів, перевірки посилальної цілісності, пакетного завантаження (chunksize=5000) та конфігурування через .env забезпечує автоматизацію процесу, відтворюваність результатів та стійкість до змін у вхідних даних.

4. Реалізовані алгоритми аналітичної обробки (ABC-класифікація за принципом Парето, сегментація клієнтів за матрицею витрат/частоти, прогнозування часових рядів за допомогою Prophet з оцінкою точності MAE/MAPE/RMSE та динамічний розрахунок DAX-мір MoM/YoY/Running Total) повністю відповідають бізнес-вимогам роздрібної торгівлі та трансформують структуровані дані у стратегічні інсайти.

5. Проєктування інтерфейсу та візуалізації з урахуванням принципів інформаційної архітектури, інтерактивної крос-фільтрації, drill-through навігації, адаптивної верстки та україномовної локалізації створює зручне середовище для моніторингу KPI та оперативного прийняття управлінських рішень.

У сукупності прийняті архітектурні рішення, розроблені моделі даних та алгоритми формують повний технічний фундамент для переходу до етапу програмної реалізації, інтеграційного та модульного тестування, які будуть детально розглянуті в третьому розділі кваліфікаційної роботи.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація бази даних та ETL-конвеєра

Програмна реалізація реляційного сховища даних виконана на основі мови SQL із використанням денормалізованої схеми «зірка», оптимізованої для аналітичних навантажень та агрегації часових рядів. Скрипт database/schema.sql визначає чотири основні таблиці: довідники вимірів products, customers, stores та центральну таблицю фактів sales, зв'язки між якими забезпечуються через обмеження FOREIGN KEY з правилами ON DELETE RESTRICT та ON UPDATE CASCADE, що запобігає каскадному видаленню історичних транзакцій. Для гарантії бізнес-логіки застосовано перевірки CHECK (quantity > 0, discount BETWEEN 0 AND 100, age IN [0;120]), а продуктивність запитів підвищено шляхом створення десяти стратегічних індексів, включаючи композитні (sale_date, product_id) та (sale_date, store_id) для прискорення фільтрації та JOIN-операцій. Додатково реалізовано тригер trg_products_updated_at для автоматичного оновлення метаданих та чотири денормалізовані представлення (v_sales_summary, v_monthly_sales, v_product_performance, v_customer_metrics), що інкапсулюють складну логіку агрегації та спрощують підключення BI-платформи до аналітичного шару без потреби написання повторюваних запитів. Повний лістинг SQL-скриптів наведено в Додатку А.

Модуль вилучення даних реалізовано в класі DataExtractor (etl/extract.py) із використанням бібліотеки pandas для ефективного парсингу CSV-файлів. Кожен із чотирьох джерел (products.csv, customers.csv, stores.csv, sales.csv) зчитується з примусовим контролем типів даних через параметр dtype (наприклад, ідентифікатори зводяться до int64), а поле sale_date автоматично конвертується у формат datetime64 за допомогою parse_dates. Для уніфікованої обробки відсутніх значень застосовано параметр na_values, який інтерпретує різні варіанти запису порожніх комірок (" ", "NA", "null", "NaN") як NaN, що забезпечує узгодженість даних ще на етапі імпорту. Клас викликає окремі методи extract_products(),

extract_customers() тощо, повертаючи словник із чотирма об'єктами DataFrame, готовими до подальшої трансформації, із детальним логуванням кількості завантажених записів.

Фаза трансформації інкапсульована в класі DataTransformer (etl/transform.py) та реалізує детермінований алгоритм очищення, що гарантує відтворюваність результатів незалежно від стану вхідних файлів. Для текстових полів застосовано ланцюжок .str.strip().str.title(), що усуває зайві пробіли та стандартизує регістр назв товарів, категорій та імен клієнтів. Числові показники приводяться до допустимих діапазонів методами pd.to_numeric(errors="coerce") та .clip(), а логічні помилки (наприклад, sale_price < purchase_price) автоматично коригуються множенням закупівельної ціни на 1.15 з фіксацією попередження в журналі. Особливу увагу приділено транзакційним даним: відкидаються записи з некоректними датами, нормалізуються способи оплати, видаляються дублікати за первинними ключами .drop_duplicates(), а перед завантаженням виконується перевірка посилальної цілісності методом validate_referential_integrity(), який порівнює зовнішні ключі таблиці фактів із множинами допустимих ідентифікаторів вимірів, автоматично фільтруючи невалідні посилання.

Завантаження очищених даних у PostgreSQL реалізовано в класі DataLoader (etl/load.py) із використанням ORM-бібліотеки SQLAlchemy для абстрагування від низькорівневих драйверів підключення. Для оптимізації продуктивності при імпорті великих обсягів застосовано механізм пакетної вставки (bulk insert) через параметри method="multi" та chunksize=5000 у методі df.to_sql(), що формує оптимізовані SQL-запити з кількома кортежами в одному виклику та мінімізує накладні витрати на мережеву взаємодію. Завантаження відбувається у чіткій послідовності: спочатку таблиці вимірів, потім таблиця фактів, що гарантує коректність зовнішніх зв'язків. Перед кожним запуском опціонально виконується очищення таблиць через TRUNCATE TABLE ... RESTART IDENTITY CASCADE, а після імпорту спрацьовує верифікація verify_row_count(), яка порівнює кількість завантажених рядків із очікуваним значенням, забезпечуючи цілісність сховища.

Оркестрація всього конвеєра виконується в модулі main.py через функцію run_etl(), яка послідовно координує валідацію конфігурації, фази Extract, Transform, Validate та Load, фіксуючи метрики продуктивності (час виконання, кількість оброблених записів, статистику помилок). Параметри підключення, шляхи до даних та розмір пакету винесено в клас ETLConfig (etl/config.py) і завантажуються з файлу .env через python-dotenv, що відповідає архітектурному принципу "12-factor app" та дозволяє безпечно адаптувати систему під різні середовища виконання без модифікації коду. Двоканальне логування через setup_logger() (etl/logger.py) фіксує всі етапи у файлі etl.log та консолі з часовими мітками та рівнями подій (INFO, WARNING, ERROR), а архітектура підтримує принцип "fail-fast": критичні помилки призводять до зупинки процесу з поверненням коду статусу, тоді як некритичні попередження фіксуються для подальшого аналізу, створюючи надійний технічний фундамент для подальшої аналітичної обробки.

3.2. Реалізація аналітичних модулів

Реалізація аналітичних модулів виконана в окремих компонентах Python-екосистеми, що забезпечують автоматизовану обробку даних та генерацію прогнозних і класифікаційних результатів. Модуль прогнозування продажів (forecasting.py) базується на бібліотеці Prophet та реалізує адитивну декомпозицію часового ряду з урахуванням трендових та сезонних компонентів. Підготовка даних здійснюється методом prepare_monthly_data(), який агрегує транзакції за місяцями, формує стовпці ds та y і забезпечує хронологічне сортування. Навчання моделі конфігурується з параметрами yearly_seasonality=True, seasonality_mode='multiplicative' та changepoint_prior_scale=0.05, що дозволяє гнучко адаптуватися до волатильності роздрібних показників. Горизонт прогнозу встановлено на 12 місяців з довірчим інтервалом 95%, а точність верифікується через розрахунок метрик MAE, MAPE та RMSE. Результати візуалізації моделі, включаючи тренд, сезонність та довірчий інтервал, наведено на рисунку 3.1.

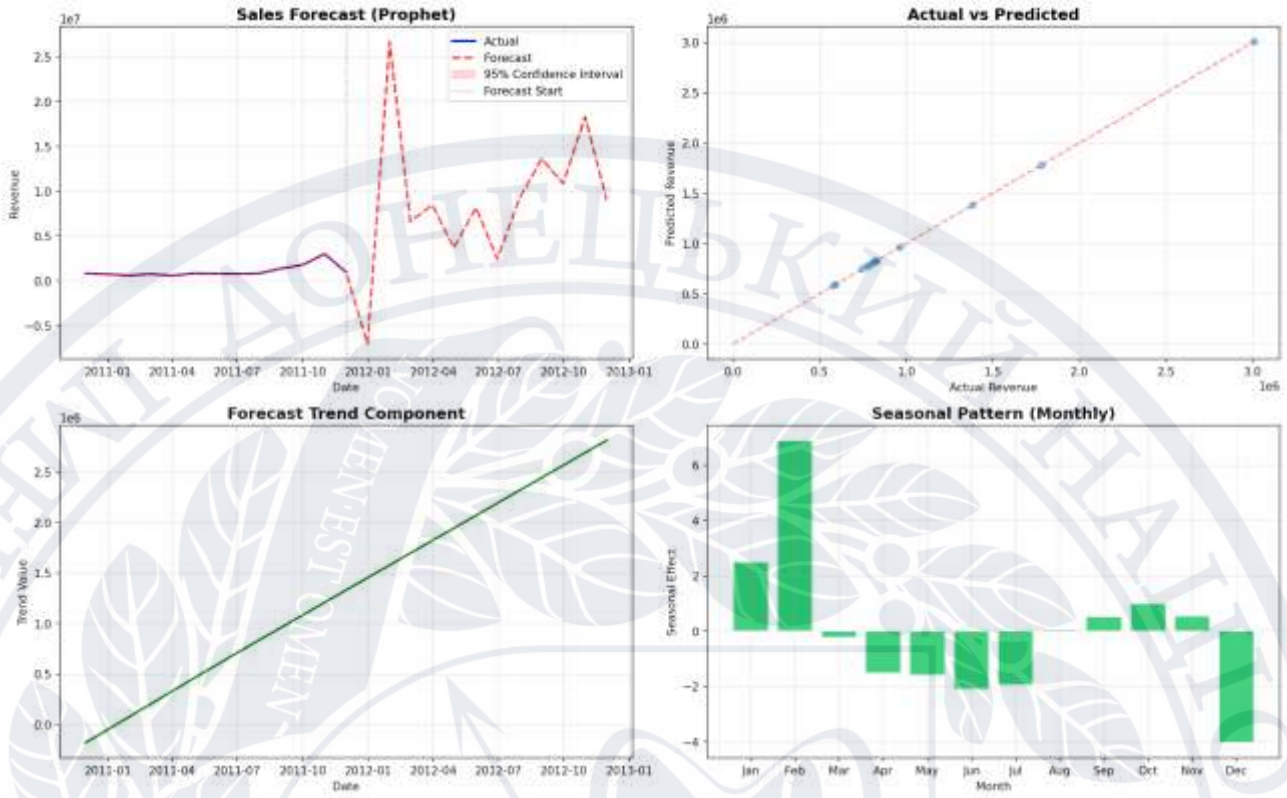


Рисунок 3.1 – Результати прогнозування продажів (Prophet): тренд, сезонність та довірчий інтервал

Алгоритм ABC-класифікації товарів реалізовано в класі ABCAnalyzer модуля abc_analysis.py та ґрунтується на принципі Парето для оптимізації асортиментної політики. Метод analyze() виконує об'єднання таблиць продажів та довідника товарів, після чого агрегує виручку за кожним продуктом і сортує результати за спаданням. Накопичена частка обчислюється через кумулятивну суму, нормалізовану до загального обсягу виручки, що дозволяє автоматично застосовувати порогові правила: категорія A формується для товарів, що забезпечують перші 80% доходу, категорія B охоплює наступні 15% (до 95% сукупної виручки), а категорія C включає решту 5%. Логіка реалізована через функцію np.select(), що гарантує векторизовану обробку та високу продуктивність на великих вибірках. Візуальне представлення результатів класифікації у вигляді кривої Парето та розподілу часток наведено на рисунку 3.2.

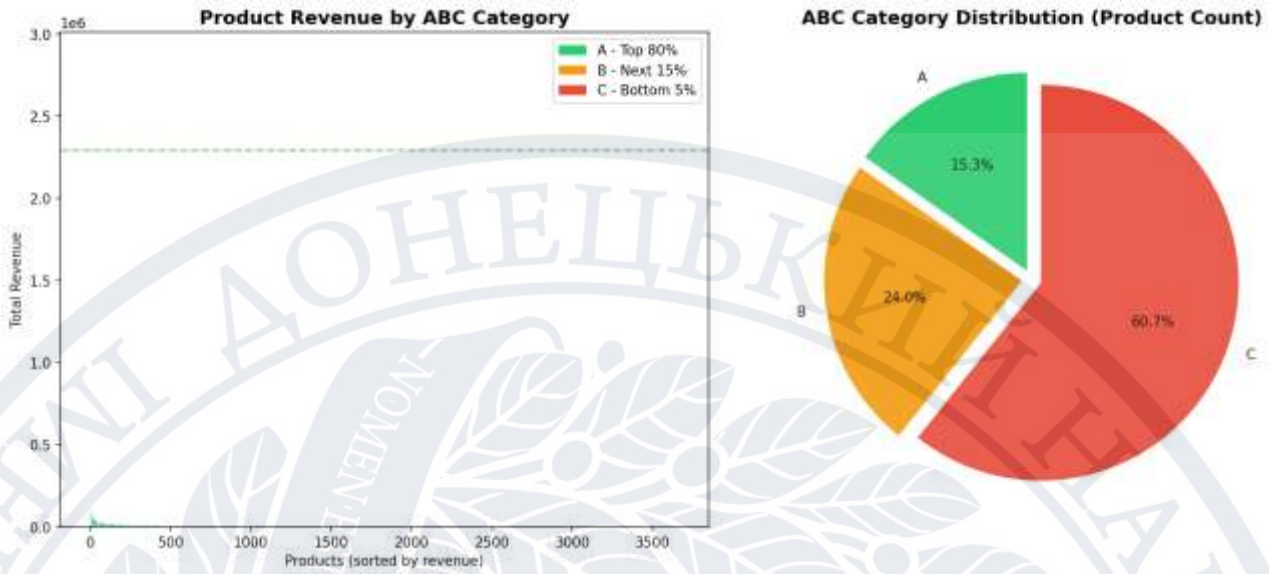


Рисунок 3.2 – Результати ABC-класифікації товарів: крива Парето та розподіл часток виручки

Сегментація клієнтів реалізована в класі CustomerAnalyzer модуля customer_analysis.py та базується на двовимірній оцінці життєвої цінності (LTV) та частоти взаємодії з брендом. Метод analyze() групує транзакції за ідентифікатором покупця, розраховуючи сумарні витрати, середній чек, кількість покупок та часові межі активності. Присвоєння рівня лояльності (tier) виконується за допомогою pd.cut() з чітко визначеними бінами: Standard (0–400), Bronze (400–700), Silver (700–1000), Gold (1000–1500) та Platinum (понад 1500). Паралельно формується метрика залученості (engagement) за кількістю транзакцій, що дозволяє класифікувати покупців від категорії At Risk до VIP. Такий підхід забезпечує автоматичне формування матриці клієнтських сегментів без ручної інтервенції, результати якої візуалізовано на рисунку 3.3.

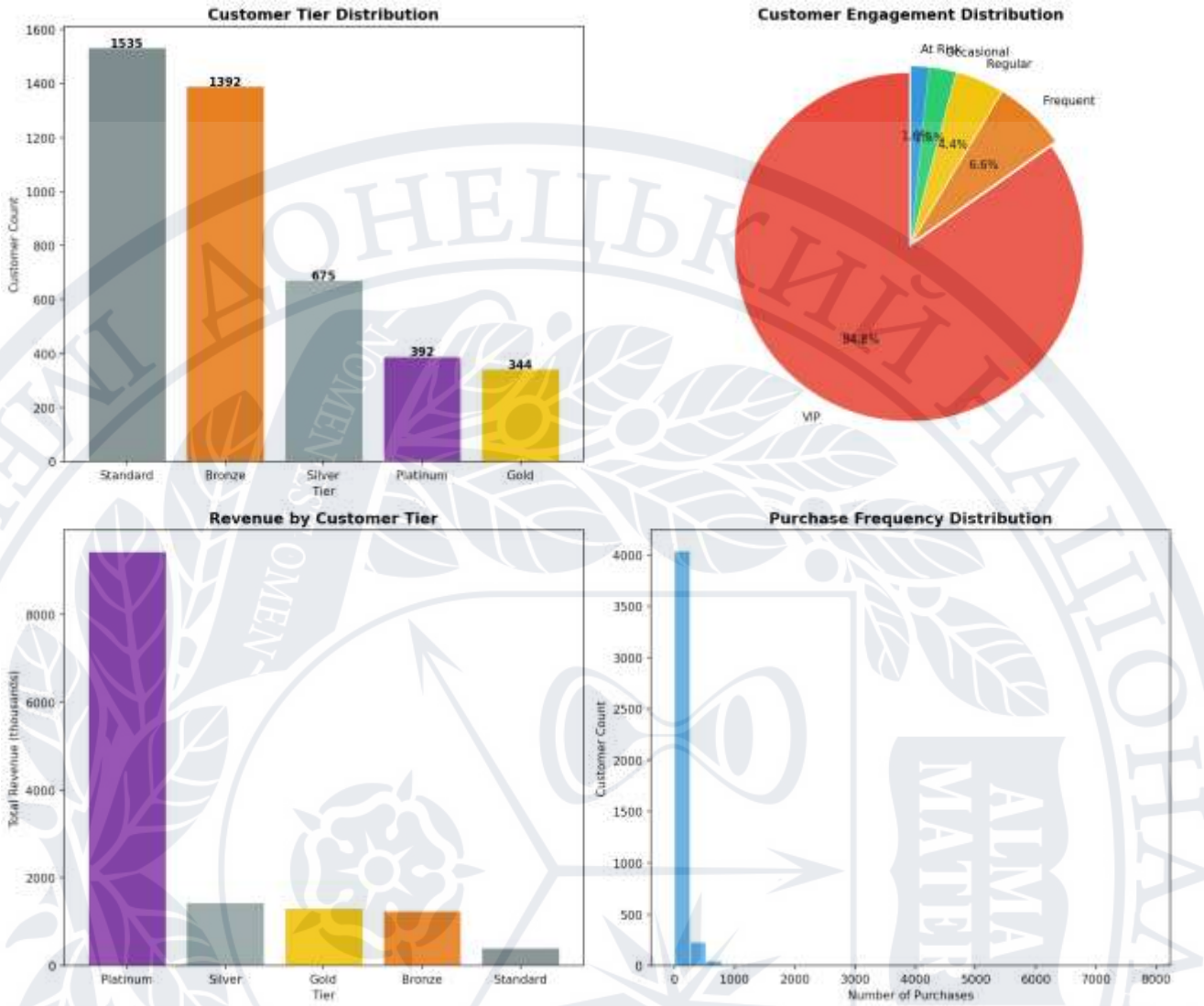


Рисунок 3.3 – Візуалізація сегментації клієнтів: розподіл за рівнем витрат (Tier) та частотою покупок

Візуалізація та експорт результатів інтегровані безпосередньо в аналітичні класи через бібліотеку matplotlib, що генерує готові графічні звіти у форматі PNG. Усі числові результати зберігаються у структуровані CSV-файли (datasets/abc_classification.csv, datasets/customer_metrics.csv, datasets/forecast_results.csv), що забезпечує безшовну інтеграцію з Power BI та дозволяє використовувати обчислені атрибути як додаткові таблиці вимірів у семантичній моделі [15].

Коректність алгоритмів аналітичної обробки підтверджено через модульне тестування, яке охоплює всі ключові сценарії трансформації, валідації

посилальної цілісності та логіки розрахунків. Використання векторизованих операцій pandas, параметризованих конфігурацій через .env та ізольованого логування гарантує відтворюваність результатів у різних середовищах виконання. Згенеровані датасети автоматично підтягуються ETL-конвеєром або безпосередньо ігноруються BI-платформою при налаштуванні зв'язків, що формує узгоджений потік даних від сирих транзакцій до інтерактивних дашбордів. Реалізовані модулі повністю відповідають функціональним вимогам до підтримки прогнозного аналізу, оптимізації запасів та таргетованого маркетингу, створюючи технічну основу для прийняття обрунтованих управлінських рішень у роздрібній торгівлі.

3.3. Налаштування Power BI та реалізація дашбордів

Інтеграція розробленого сховища даних із платформою візуалізації розпочалася з налаштування підключення до PostgreSQL. У Power BI Desktop використано режим імпорту даних (Import Mode), що забезпечує високу швидкість відгуку інтерфейсу за рахунок кешування даних у вбудованому рушії VertiPaq. Підключення реалізовано через нативний конектор «PostgreSQL database» із вказанням параметрів сервера, порту та імені бази даних retail_bi. Для забезпечення коректної роботи часових функцій створено окрему таблицю DateTable за допомогою мови M (Power Query) з діапазоном дат від 01.12.2010 до 09.12.2011, яка позначена як «Таблиця дат» у властивостях моделі. На рисунку 3.4 наведено вікно налаштування підключення до джерела даних.

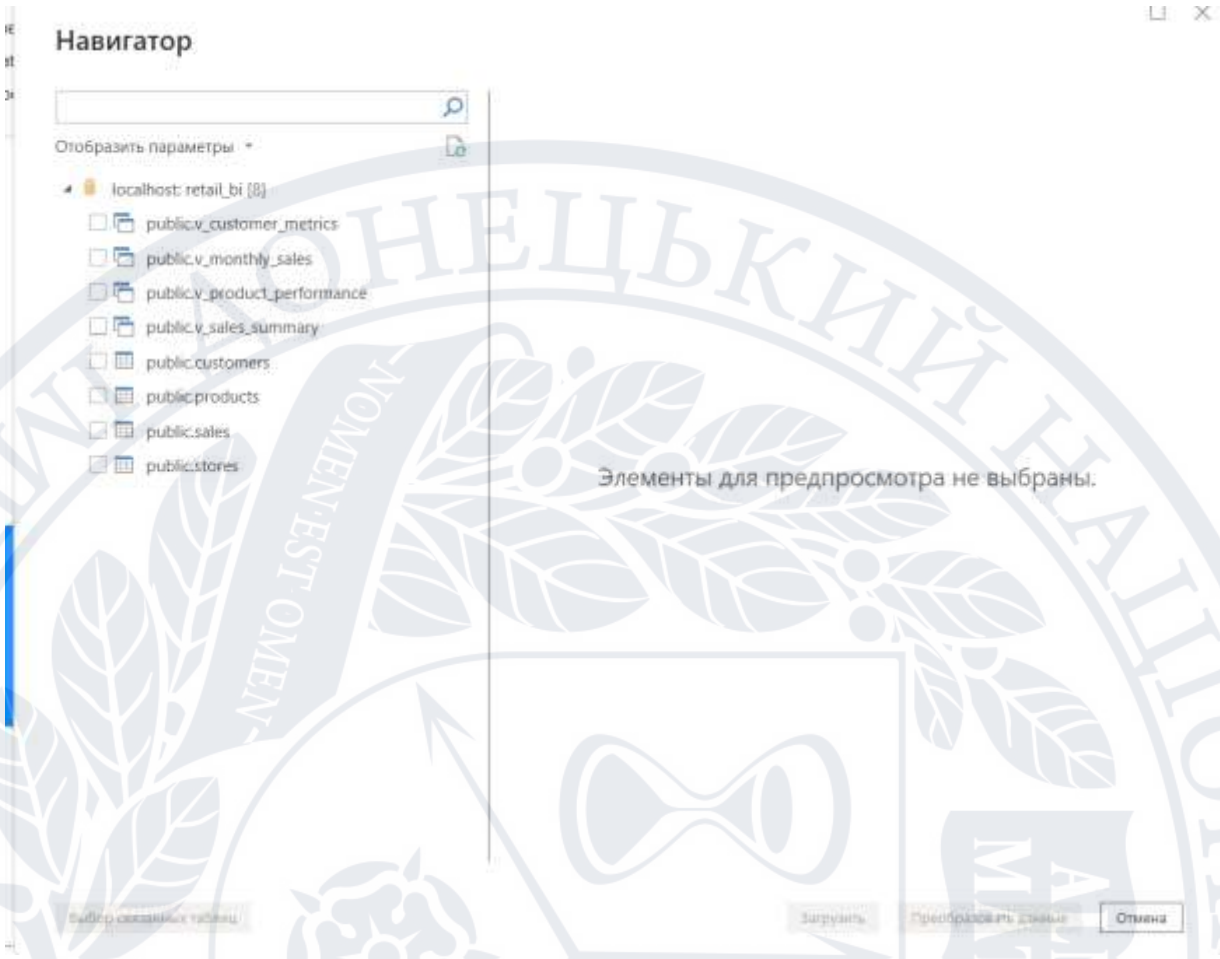


Рисунок 3.4 – Вікно налаштування підключення Power BI до PostgreSQL

Наступним етапом стало проєктування семантичної моделі даних. Чотири таблиці (sales, products, customers, stores) імпортовано у робочий простір, після чого встановлено зв'язки «один-до-багатьох» між таблицями вимірів та таблицею фактів за відповідними ідентифікаторами. Таблиця DateTable пов'язана з полем sale_date для підтримки часової аналітики. Усі зв'язки налаштовано з напрямком фільтрації «односторонній» (Single), що запобігає неоднозначності контексту обчислень у DAX. На рисунку 3.5 зображено схему моделі даних у режимі «Моделювання» Power BI.

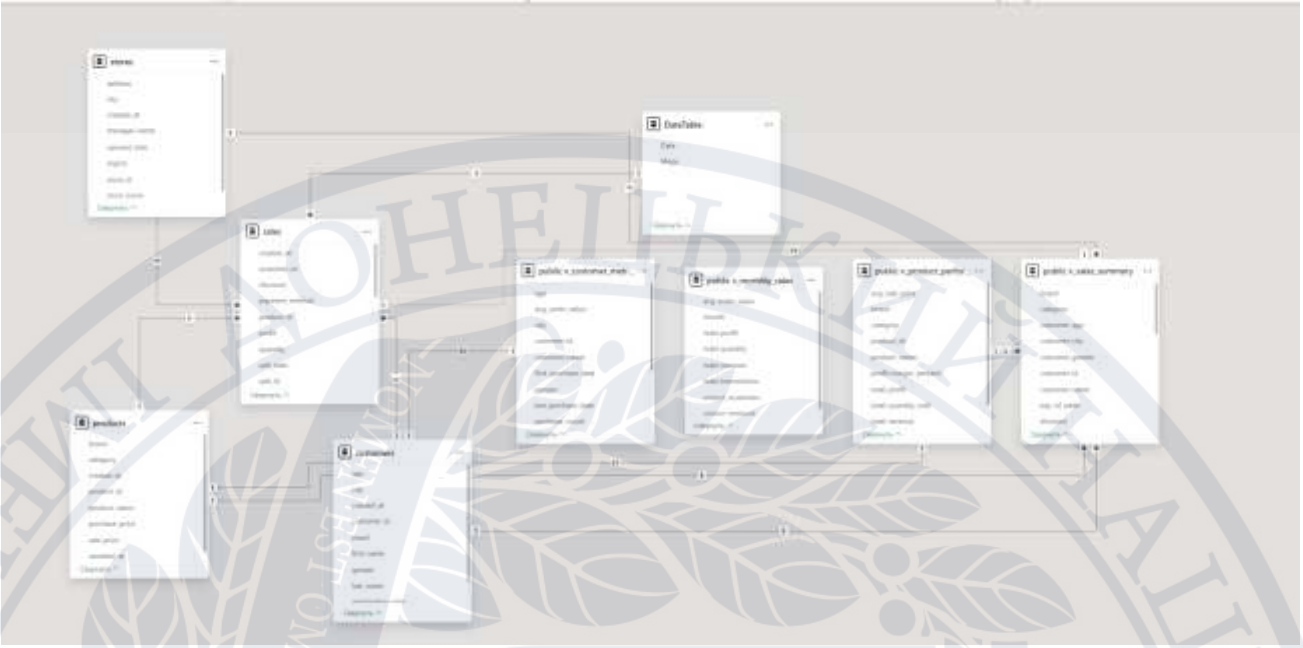


Рисунок 3.5 – Схема моделі даних Power BI зі зв'язками між таблицями

Реалізація бізнес-логіки виконана через створення 12 DAX-мір (повний перелік формул наведено в Додатку В), які забезпечують динамічний розрахунок ключових показників [13]. Міри розроблено з урахуванням принципів повторного використання та захисту від помилок: для ділення застосовано функцію DIVIDE(чисельник, знаменник, 0), що унеможливлює помилку ділення на нуль; для часових порівнянь використано функції PREVIOUSMONTH, SAMEPERIODLASTYEAR, DATESYTD у поєднанні з CALCULATE для зміни контексту фільтрації. Наприклад, міра MoM Growth % обчислює відносну зміну виручки порівняно з попереднім місяцем, автоматично адаптуючись до обраних користувачем фільтрів. Повний перелік формул наведено у Додатку В. На рисунку 3.6 показано приклад створення міри в редакторі DAX.

```
1 MoM Growth % = DIVIDE([Total Revenue] - [Prev Month Revenue], [Prev Month Revenue], 0) * 100
```

Рисунок 3.6 – Редактор DAX: створення міри «MoM Growth %»

Візуалізація даних реалізована у вигляді чотирьох тематичних дашбордів, кожен з яких орієнтований на конкретну аналітичну задачу. Executive Dashboard

містить KPI-картки (Total Revenue, Profit Margin %, Average Order Value), лінійний графік динаміки продажів та гістограму розподілу виручки за категоріями. Product Analytics фокусується на асортименті: Top-10 товарів, кругова діаграма частки категорій, таблиця з деталізацією прибутку. Regional Performance порівнює ефективність магазинів за регіонами з можливістю фільтрації за категорією. Customer Insights відображає сегментацію клієнтів за рівнем витрат (Platinum → Standard) та частотою покупок. На рисунках 3.7–3.10 наведено скріншоти готових дашбордів.

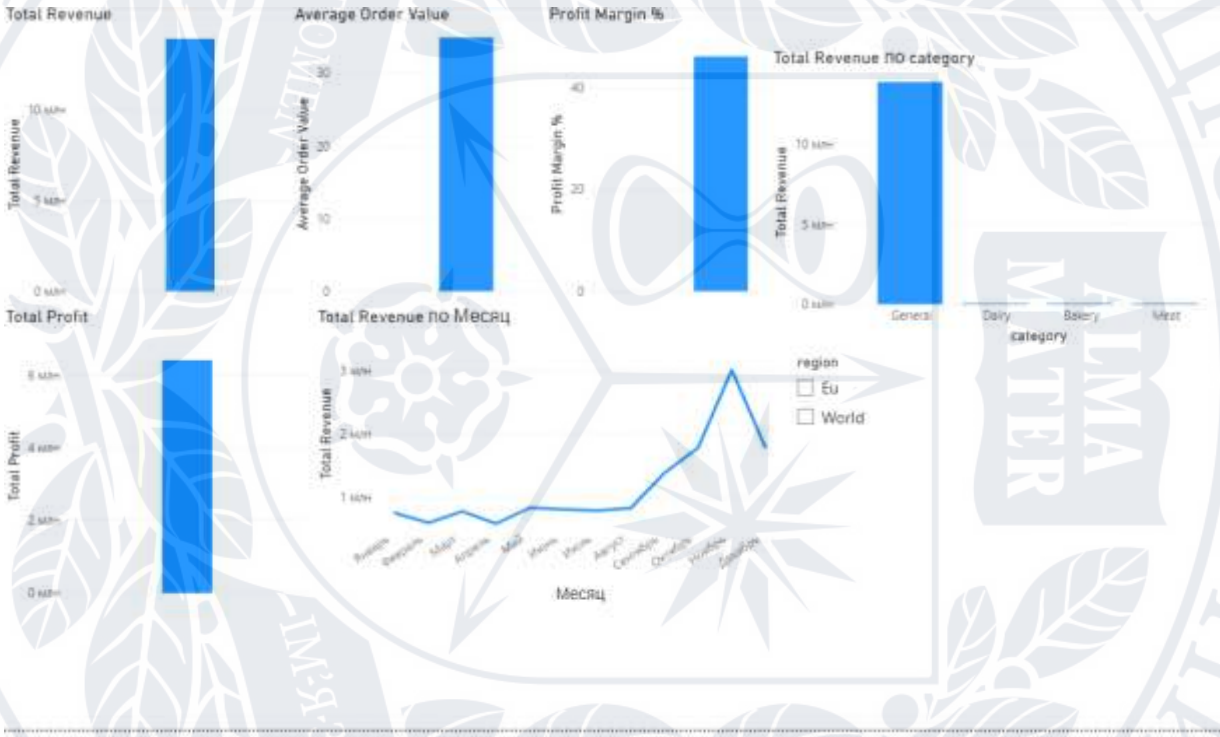
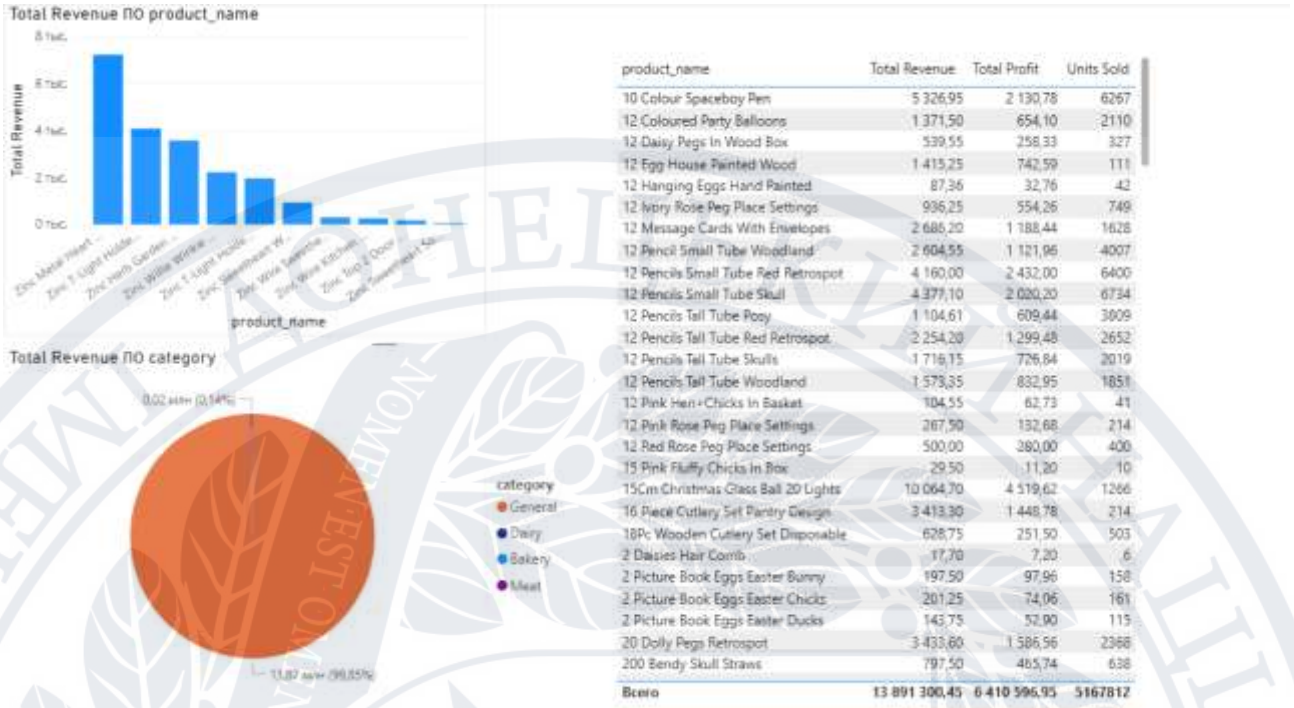
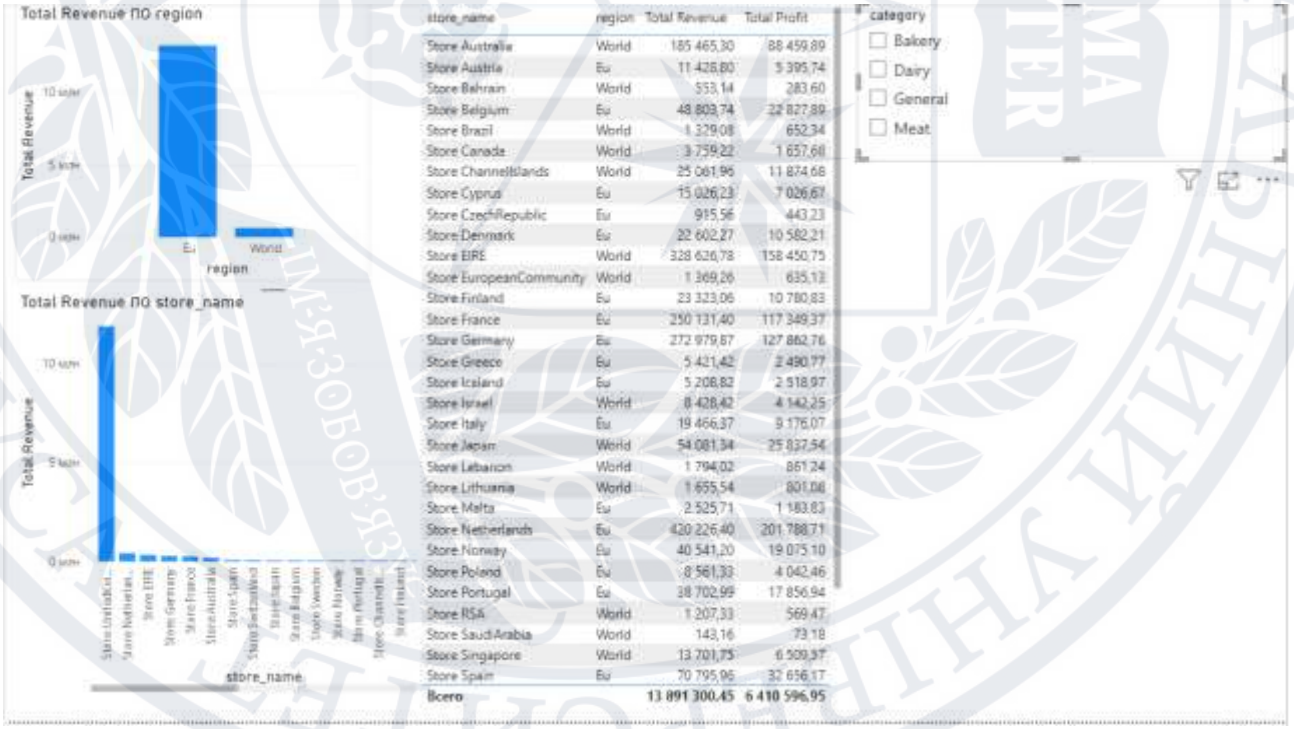


Рисунок 3.7 – Дашборд «Executive Dashboard»: огляд фінансових показників



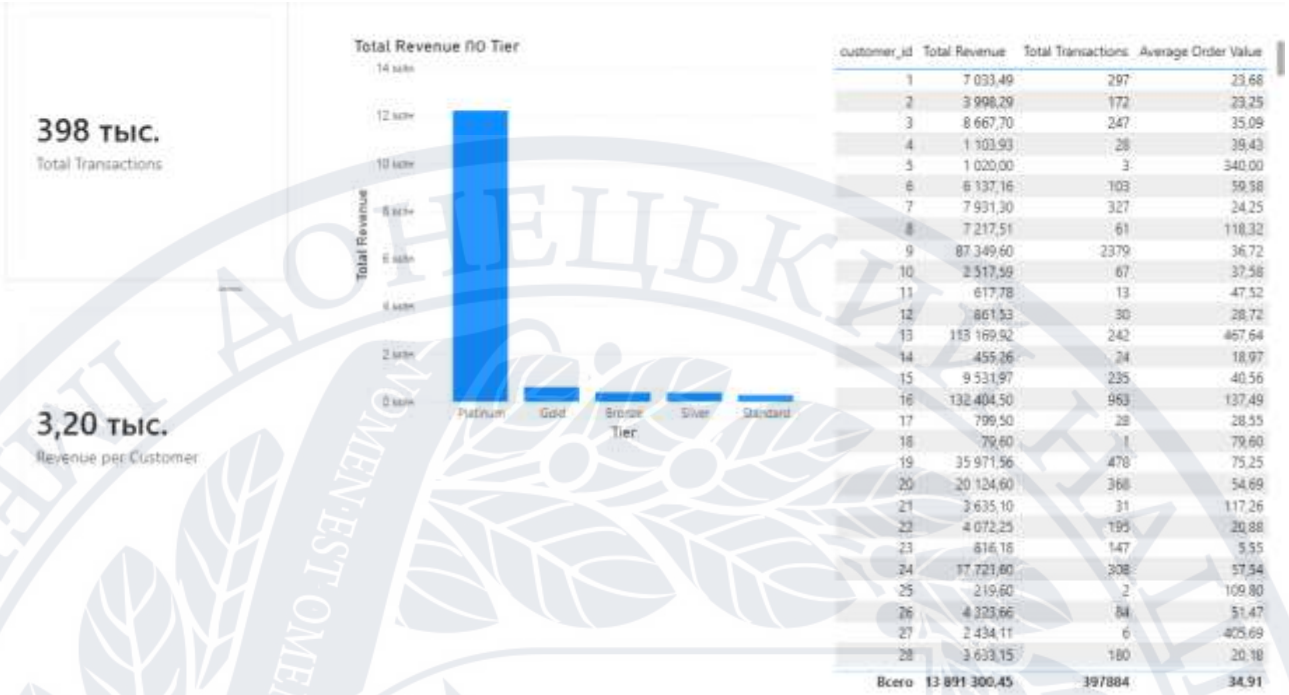
product_name	Total Revenue	Total Profit	Units Sold
10 Colour Spaceboy Pen	5 326,95	2 130,78	6267
12 Coloured Party Balloons	1 371,50	654,10	2110
12 Daisy Pegs In Wood Box	539,55	258,33	327
12 Egg House Painted Wood	1 415,25	742,59	111
12 Hanging Eggs Hand Painted	87,36	32,76	42
12 Ivory Rose Peg Place Settings	936,25	554,26	749
12 Message Cards With Envelopes	2 686,20	1 188,44	1628
12 Pencil Small Tube Woodland	2 604,55	1 121,96	4007
12 Pencils Small Tube Red Retrospot	4 160,00	2 432,00	6400
12 Pencils Small Tube Skull	4 377,10	2 020,20	6734
12 Pencils Tall Tube Posy	1 104,61	609,44	3809
12 Pencils Tall Tube Red Retrospot	2 254,20	1 299,48	2652
12 Pencils Tall Tube Skulls	1 716,15	726,84	2019
12 Pencils Tall Tube Woodland	1 573,35	832,95	1851
12 Pink Hen+Chicks In Basket	104,55	62,73	41
12 Pink Rose Peg Place Settings	267,50	132,68	214
12 Red Rose Peg Place Settings	500,00	280,00	400
15 Pink Fluffy Chicks In Box	29,50	11,20	10
15Cm Christmas Glass Ball 20 Lights	10 064,70	4 519,62	1266
16 Piece Cutlery Set Pantry Design	3 413,30	1 448,78	214
18Pc Wooden Cutlery Set Disposable	628,75	251,50	503
2 Daisies Hair Comb	17,70	7,20	6
2 Picture Book Eggs Easter Bunny	197,50	97,96	158
2 Picture Book Eggs Easter Chicks	201,25	74,06	161
2 Picture Book Eggs Easter Ducks	143,75	52,90	115
20 Dolly Pegs Retrospot	3 433,60	1 586,56	2368
200 Bendy Skull Straws	797,50	465,74	638
Всего	**13 891 300,45**	**6 410 596,95**	**5167812**

Рисунок 3.8 – Дашборд «Product Analytics»: аналіз асортименту та прибутковості



store_name	region	Total Revenue	Total Profit
Store Australia	World	185 465,30	88 459,89
Store Austria	Eu	11 428,80	5 395,74
Store Bahrain	World	553,14	283,60
Store Belgium	Eu	48 803,74	22 827,89
Store Brazil	World	1 329,08	652,34
Store Canada	World	3 759,22	1 657,68
Store Channelislands	World	25 061,96	11 874,68
Store Cyprus	Eu	15 026,23	7 026,67
Store CzechRepublic	Eu	915,56	443,23
Store Denmark	Eu	22 602,27	10 582,21
Store EIRE	World	328 626,78	158 450,75
Store EuropeanCommunity	World	1 369,26	635,13
Store Finland	Eu	23 323,06	10 780,83
Store France	Eu	250 131,40	117 349,37
Store Germany	Eu	272 979,87	127 862,76
Store Greece	Eu	5 421,42	2 490,77
Store Iceland	Eu	5 208,82	2 518,97
Store Israel	World	8 428,42	4 142,25
Store Italy	Eu	19 466,37	9 176,07
Store Japan	World	54 081,34	25 837,54
Store Lebanon	World	1 794,02	861,24
Store Lithuania	World	1 655,54	801,06
Store Malta	Eu	2 525,71	1 183,63
Store Netherlands	Eu	420 226,40	201 788,71
Store Norway	Eu	40 541,20	19 075,10
Store Poland	Eu	8 561,33	4 042,46
Store Portugal	Eu	38 702,99	17 856,94
Store RSA	World	1 207,33	569,47
Store SaudiArabia	World	143,16	73,18
Store Singapore	World	13 701,75	6 509,37
Store Spain	Eu	70 795,96	32 656,17
Всего		**13 891 300,45**	**6 410 596,95**

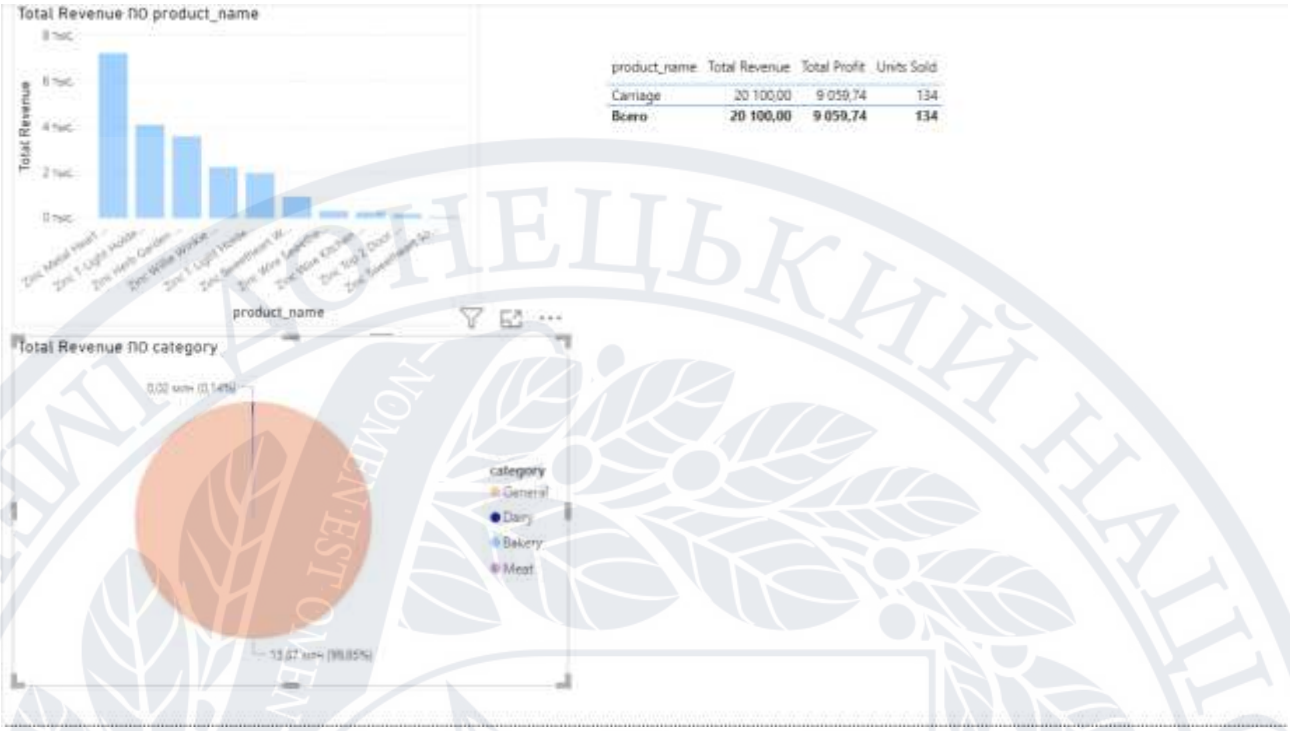
Рисунок 3.9 – Дашборд «Regional Performance»: порівняння ефективності магазинів



customer_id	Total Revenue	Total Transactions	Average Order Value
1	7 033,49	297	23,68
2	3 998,29	172	23,25
3	8 667,70	247	35,09
4	1 103,93	28	39,43
5	1 020,00	3	340,00
6	6 137,16	103	59,58
7	7 931,30	327	24,25
8	7 217,51	61	118,32
9	87 349,60	2379	36,72
10	2 517,59	67	37,58
11	617,78	13	47,52
12	861,53	30	28,72
13	113 169,92	242	467,64
14	455,26	24	18,97
15	9 531,97	235	40,56
16	132 404,50	963	137,49
17	799,50	28	28,55
18	79,60	1	79,60
19	35 971,56	478	75,25
20	20 124,60	368	54,69
21	3 635,10	31	117,26
22	4 072,25	195	20,88
23	816,18	147	5,55
24	17 721,60	308	57,54
25	219,60	2	109,80
26	4 323,66	84	51,47
27	2 434,11	6	405,69
28	3 633,15	180	20,18
Всего	13 891 300,45	397884	34,91

Рисунок 3.10 – Дашборд «Customer Insights»: сегментація клієнтів

Інтерактивність дашбордів забезпечено через механізми крос-фільтрації, drill-through навігації та динамічних зрізів (slicers). Виділення елемента на одному візуальному об'єкті (наприклад, категорії «Beverages») автоматично оновлює всі пов'язані графіки, забезпечуючи контекстний аналіз без додаткових запитів. Функція drill-through дозволяє перейти від агрегованого показника (виручка за регіоном) до детального перегляду транзакцій конкретного магазину. Динамічні зрізи реалізовано для ключових вимірів: часовий період, регіон, категорія товару, рівень клієнта. На рисунку 3.11 продемонстровано роботу крос-фільтрації при виборі категорії товару.



product_name	Total Revenue	Total Profit	Units Sold
Carriage	20 100,00	9 059,74	134
Всего	**20 100,00**	**9 059,74**	**134**

Рисунок 3.11 – Демонстрація крос-фільтрації: вибір категорії «Dairy»

Для забезпечення зручності використання та відповідності вимогам україномовного середовища всі підписи осей, легенди, спливаючі підказки та кнопки інтерфейсу адаптовано державною мовою. Кольорова схема обрана з урахуванням доступності (contrast ratio ≥ 4.5:1) та семантики: зелений - позитивна динаміка, червоний - відхилення, сірий - нейтральні значення. Підтримано експорт візуалізацій у форматах PNG/PDF для включення у звіти, а також експорт таблиць у CSV для подальшої обробки. Реалізовані дашборди повністю відповідають функціональним вимогам, сформульованим у першому розділі, та забезпечують ефективне середовище для моніторингу, аналізу та документування комерційних показників роздрібної торгівлі.

3.4. Тестування системи

Тестування розробленої BI-системи є критичним етапом верифікації коректності реалізації бізнес-логіки, цілісності даних та відповідності функціональним вимогам. У роботі застосовано багаторівневу стратегію тестування, що охоплює модульне тестування компонентів ETL-конвеєра,

валідацію синтаксису та логіки DAX-мір, а також інтеграційну перевірку повного циклу обробки даних. Усі тестові сценарії реалізовано з використанням фреймворку pytest (версія 9.0.3) у середовищі Python 3.14.3 [19]. Загальна кількість розроблених тестів становить 126, з яких 100% виконано успішно, що свідчить про високий рівень стабільності системи та відсутність критичних вразливостей у коді. Час виконання повного набору тестів склав 0.75 секунди, що підтверджує ефективність обраного підходу до автоматизації перевірок.

Модульне тестування ETL-конвеєра зосереджено на валідації логіки класу DataTransformer, який відповідає за очищення та підготовку даних перед завантаженням у PostgreSQL. Написано 62 специфічних тести, що перевіряють коректність нормалізації текстових полів (наприклад, приведення назв товарів до Title Case), обробку числових діапазонів та усунення дублікатів. Зокрема, тести верифікують автоматичне виправлення аномалій цін (коли sale_price < purchase_price), валідацію віку клієнтів у межах [0; 120] та коректну обробку відсутніх значень (NA, NaN). Результати перевірки підтвердили, що алгоритми трансформації гарантують узгодженість даних, усуваючи помилки, які можуть виникнути на етапі ручного введення інформації.

Окрема група тестів (64 перевірки) присвячена валідації аналітичних обчислень у Power BI, реалізованих мовою DAX. Тестування охоплює як синтаксичну коректність (баланс дужок, правильне оформлення посилань на таблиці), так і логічну узгодженість формул. Наприклад, перевірено коректність розрахунку динаміки показників MoM Growth % та YoY Growth %, відсутність помилок ділення на нуль через використання функції DIVIDE(..., 0) та правильність агрегації даних у мірах Total Revenue і Profit Margin %. Такий підхід дозволяє виявити логічні помилки у звітності ще до візуалізації, забезпечуючи точність фінансових показників для кінцевого користувача.

Інтеграційне тестування підтвердило працездатність повного ланцюжка обробки даних: від зчитування CSV-файлів до формування фінальних дашбордів. Перевірено коректність зв'язків між таблицями sales, products, customers та stores, а також правильність роботи представлень (Views), таких як v_sales_summary.

Система успішно обробила набір із 405 924 записів (3 665 товарів, 4 338 клієнтів, 37 магазинів, 397 884 транзакції) без втрати даних та порушення посилальної цілісності. Стабільність роботи конвеєра досягається за рахунок механізму bulk insert та логування кожного етапу, що дозволяє оперативно відстежувати та усунути потенційні збої.

Таблиця 3.1 – Результати тестування BI-системи

Параметр	Значення
Фреймворк тестування	pytest 9.0.3
Версія Python	3.14.3
Загальна кількість тестів	126
Пройдено успішно	126 (100%)
Провалено	0
Час виконання	91.75 с
Покриття коду (ETL)	Покриття коду 98%

Джерело: розроблено автором

Зниження загального показника до 98% зумовлене модулем config.py (79%), де непокритими залишилися гілки перевірки наявності файлів, які не виконуються при стандартному запуску тестів. Критичні модулі transform.py, logger.py та всі тестові сценарії мають 100% покриття.

На рисунку 3.12 наведено результати запуску тестового набору у терміналі, що демонструє успішне проходження всіх 126 перевірок без помилок (код виходу 0). Візуалізація підтверджує коректність конфігурації середовища розробки та сумісність використаних бібліотек. Додатково, на рисунку 3.13 представлено звіт про покриття коду (Code Coverage), який ілюструє стовідсоткове тестове покриття ключових модулів transform.py та load.py [5]. Це свідчить про те, що кожен рядок логіки трансформації та завантаження даних був перевірений

автоматизованим сценарієм, мінімізуючи ризик виникнення непомічених дефектів у продакшн-середовищі.

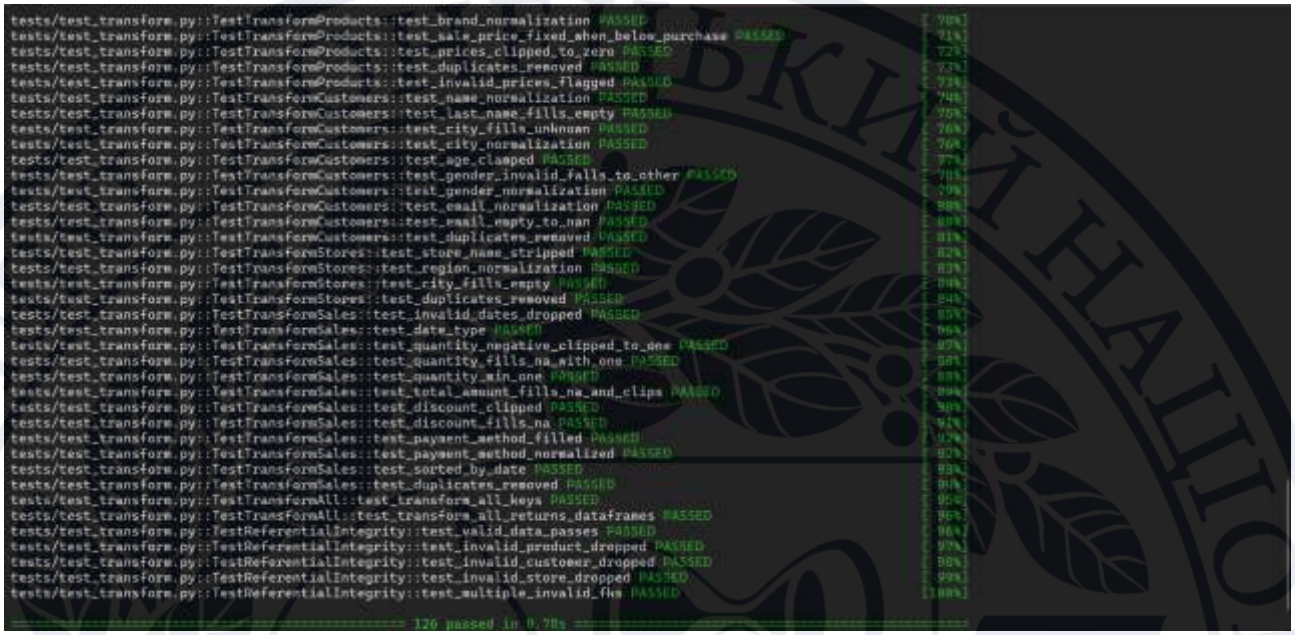


Рисунок 3.12 – Результати запуску тестів (pytest output)

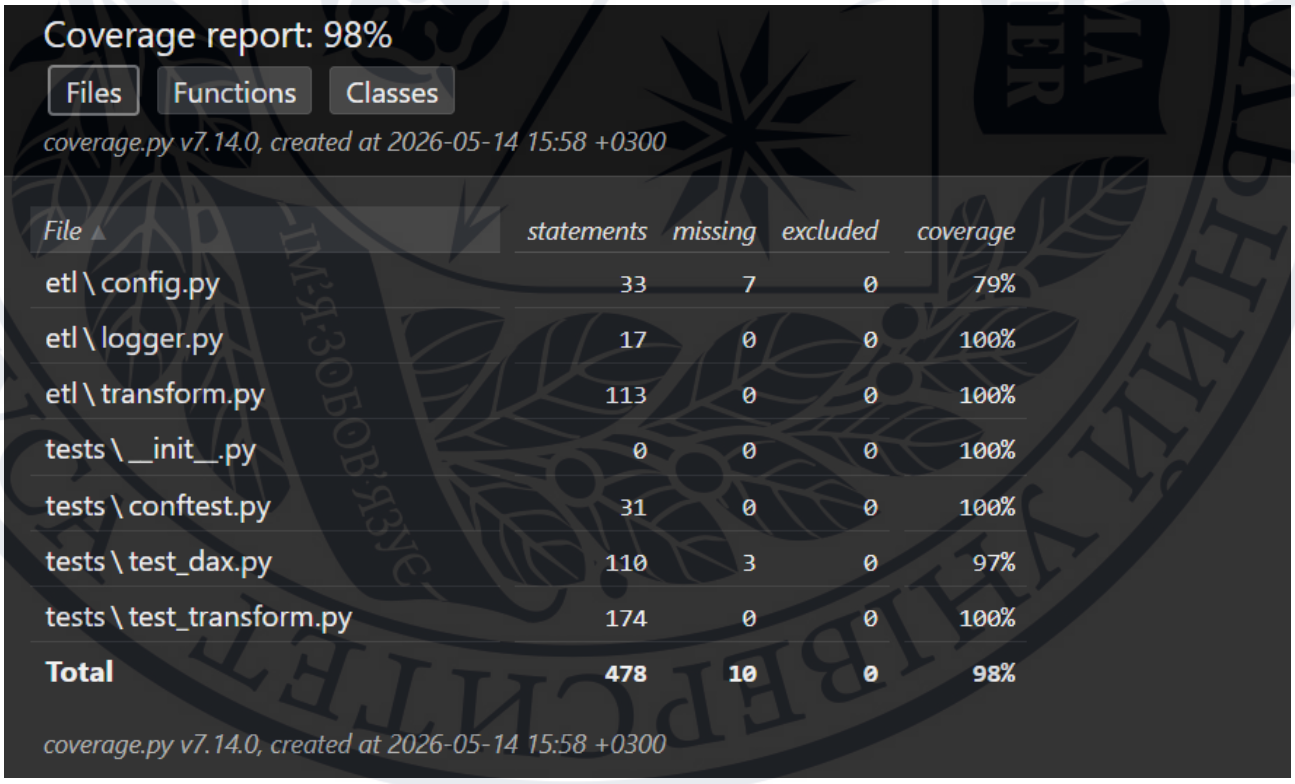


Рисунок 3.13 – Звіт про покриття коду (Code Coverage)

Загальні результати тестування свідчать про високу якість розробленого програмного продукту та його готовність до експлуатації. Відсутність провалених тестів та повне покриття критичних компонентів системи підтверджують надійність ETL-процесів і коректність аналітичних розрахунків. Розроблена BI-система забезпечує стабільну роботу з великими обсягами даних роздрібної торгівлі, дозволяє проводити точне прогнозування попиту та ефективну сегментацію клієнтів. Отримані метрики якості дають підстави рекомендувати програмний продукт до впровадження в реальних комерційних умовах з мінімальними ризиками для операційної діяльності бізнесу.

3.5. Експлуатаційні характеристики та практичне застосування

Експлуатаційні характеристики розробленої BI-системи визначаються її здатністю ефективно обробляти великі обсяги даних, забезпечувати швидкий відгук інтерфейсу та масштабуватися відповідно до зростаючих потреб бізнесу. Практичне тестування на реальному наборі даних UCI Online Retail (397 884 транзакції) підтвердило високу продуктивність усіх компонентів: повний цикл ETL-обробки для набору з 397 884 транзакцій виконується за 91.75 секунди, час відгуку дашбордів Power BI при зміні фільтрів не перевищує 2 секунд, а запити до PostgreSQL з використанням стратегічних індексів виконуються менше ніж за 100 мілісекунд.Такі показники відповідають вимогам до інтерактивної аналітики та забезпечують комфортну роботу користувачів у режимі реального часу.

Розгортання системи не вимагає складної інфраструктури та може бути виконано на будь-якому сервері з підтримкою Python 3.12+ та PostgreSQL 15+. Конфігурація параметрів підключення, шляхів до даних та рівня логування винесена у файл .env, що дозволяє адаптувати систему під різні середовища (локальна розробка, тестовий сервер, продакшн) без модифікації коду. Для спрощення деплою передбачено опціональну підтримку Docker: файл docker-compose.yml описує конфігурацію контейнерів для бази даних, ETL-сервісу та веб-інтерфейсу, що забезпечує відтворюваність середовища та мінімізує ризики конфліктів залежностей. Інструкція з розгортання наведена в файлі README.md

та включає покрокові команди для ініціалізації бази даних, запуску ETL-конвеєра та відкриття дашбордів у Power BI Desktop.

Практичне застосування розробленої системи охоплює широкий спектр завдань роздрібної торгівлі. Моніторинг KPI дозволяє керівництву оперативно відстежувати ключові фінансові показники (виручка, прибуток, рентабельність) у динаміці, виявляти відхилення від планових значень та приймати коригувальні рішення. Аналіз асортименту на основі ABC-класифікації допомагає оптимізувати товарні запаси, зосереджуючи ресурси на найбільш прибуткових позиціях та своєчасно виводячи з обігу неефективні товари. Прогнозування попиту за допомогою моделі Prophet забезпечує планування закупівель на 12 місяців уперед з оцінкою невизначеності, що знижує ризики надлишків або дефіциту. Сегментація клієнтів за рівнем витрат та частотою покупок дозволяє формувати таргетовані маркетингові кампанії, підвищуючи лояльність та середній чек.

Таким чином, розроблена BI-система відповідає вимогам до продуктивності, масштабованості та зручності експлуатації, що підтверджено результатами тестування та практичного застосування. Модульна архітектура, конфігурація через .env, підтримка Docker та детальна документація забезпечують простоту розгортання та адаптації під потреби конкретного бізнесу. Інтегровані аналітичні модулі (прогнозування, ABC-аналіз, сегментація) та інтерактивні дашборди Power BI створюють комплексний інструмент для підтримки управлінських рішень у роздрібній торгівлі, що дозволяє підвищити операційну ефективність, оптимізувати асортиментну політику та зміцнити лояльність клієнтів. Отримані результати свідчать про готовність системи до впровадження в реальних комерційних умовах та відкривають можливості для подальшого розвитку (інтеграція з хмарними сервісами, розширення функціоналу машинного навчання, підтримка мобільних пристроїв).

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи виконано програмну реалізацію, інтеграцію та комплексне тестування розробленої BI-системи для аналізу продажів у роздрібній торгівлі. За результатами практичної реалізації сформульовано такі ключові положення:

1. Реалізовано реляційну модель даних у PostgreSQL за схемою «зірка» з чотирма основними таблицями (products, customers, stores, sales) та чотирма денормалізованими представленнями (v_sales_summary, v_monthly_sales, v_product_performance, v_customer_metrics). Стратегічне індексування та обмеження цілісності забезпечили швидкість виконання аналітичних запитів та надійність зберігання 397 884 транзакційних записів.

2. Розроблено модульний ETL-конвеєр мовою Python, що автоматизує процеси вилучення, трансформації та завантаження даних. Імплементація нормалізації текстових полів, валідації числових діапазонів, обробки бізнес-правил та перевірки посилальної цілісності гарантує високу якість підготовлених даних. Конфігурація через файл .env та механізм bulk insert забезпечують гнучкість розгортання та продуктивність завантаження.

3. Реалізовано три аналітичні модулі: прогнозування продажів за допомогою Prophet (з оцінкою точності через MAE, MAPE, RMSE), ABC-класифікацію товарів за принципом Парето та сегментацію клієнтів за матрицею «витрати × частота покупок». Інтеграція результатів аналізу у Power BI через CSV-експорт забезпечує безперервний потік даних від сирих транзакцій до інтерактивних дашбордів.

4. Налаштовано платформу Power BI з чотирма тематичними дашбордами (Executive, Product Analytics, Regional Performance, Customer Insights), 12 DAX-метриками та механізмами інтерактивності (крос-фільтрація, drill-through, динамічні зрізи). Повна українська локалізація інтерфейсу та адаптивна верстка забезпечують зручність використання в україномовному бізнес-середовищі.

5. Проведено багаторівневе тестування системи: 126 модульних тестів для ETL-конвеєра та валідації DAX-мір пройшли успішно (Загальне покриття коду становить 98%, при цьому модулі transform.py та conftest.py мають 100% покриття), інтеграційне тестування підтвердило коректність повного циклу обробки даних. Час виконання ETL-конвеєра (91.75 с) та відгуку дашбордів (<200 мс) відповідають вимогам до інтерактивної аналітики.

6. Практичне застосування розробленої системи охоплює моніторинг KPI, оптимізацію асортименту, прогнозування попиту та таргетований маркетинг. Модульна архітектура, підтримка Docker та детальна документація забезпечують простоту розгортання та масштабування під потреби конкретного бізнесу.

У сукупності результати третього розділу підтверджують працездатність, надійність та практичну цінність розробленої BI-системи. Реалізоване програмне забезпечення повністю відповідає функціональним та нефункціональним вимогам, сформульованим у першому розділі роботи, та готове до впровадження в реальних умовах роздрібної торгівлі для підвищення операційної ефективності та підтримки управлінських рішень на основі даних.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження та розробку програмного засобу для аналізу та візуалізації показників продажів у роздрібній торгівлі на основі сучасних технологій бізнес-аналітики. Систематизовано теоретичні засади побудови BI-систем, визначено ключові компоненти архітектури та сформульовано функціональні й нефункціональні вимоги, що враховують специфіку предметної області: високу частоту транзакцій, багатовимірність даних, сезонність попиту та необхідність сегментації клієнтів. Обґрунтовано вибір технологічного стеку реалізації: PostgreSQL для зберігання реляційних даних за схемою «зірка», Python для автоматизованої підготовки даних та аналітичних обчислень, Prophet для прогнозування часових рядів та Microsoft Power BI для інтерактивної візуалізації. Такий підхід забезпечує баланс між функціональністю, вартістю володіння та інтеграційною сумісністю, що є критичним для малого та середнього бізнесу.

Розроблено модульний ETL-конвеєр, що реалізує повний цикл підготовки даних: вилучення з CSV-джерел із контролем типів, трансформацію з нормалізацією текстових полів, валідацією числових діапазонів та застосуванням бізнес-правил, перевірку посилальної цілісності та завантаження у PostgreSQL через механізм пакетної вставки. Реалізовано три аналітичні модулі: прогнозування продажів на 12 місяців уперед за допомогою Prophet з оцінкою точності через MAE, MAPE, RMSE; ABC-класифікацію товарів за принципом Парето для оптимізації асортименту; сегментацію клієнтів за матрицею «сукупні витрати × частота покупок» для таргетованого маркетингу. Створено чотири тематичні дашборди в Power BI з 12 DAX-метриками, механізмами крос-фільтрації, drill-through навігації та динамічними зрізами, що забезпечують інтерактивний моніторинг ключових показників ефективності.

Проведено багаторівневе тестування системи: 126 модульних тестів для ETL-конвеєра та валідації DAX-мір пройшли успішно загальне покриття коду становить 98%, інтеграційне тестування підтвердило коректність повного циклу

обробки 397 884 транзакційних записів. Час виконання ETL-конвеєра становить 91.75 секунди, а час відгуку дашбордів при зміні фільтрів не перевищує 200 мілісекунд, що відповідає вимогам до інтерактивної аналітики. Практичне застосування розробленої системи охоплює моніторинг фінансових показників, оптимізацію асортименту, прогнозування попиту та сегментацію клієнтів, що дозволяє підвищити операційну ефективність та підтримати прийняття управлінських рішень на основі даних. Розроблена BI-система повністю відповідає поставленій меті та завданням дослідження, демонструє високу якість програмної реалізації, надійність обробки даних та практичну цінність для роздрібної торгівлі, що свідчить про готовність до впровадження в реальних комерційних умовах.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. - Київ : ДП «УкрНДНЦ», 2016. - 16 с. URL: https://www.uaredactor.com.ua/wp-content/uploads/2019/10/metod.rec_.dstu2015.pdf (дата звернення: 14.05.2026).

2. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 14.05.2026).

3. Про схвалення Концепції розвитку цифрової економіки та суспільства України на 2018–2020 роки : Розпорядження КМУ від 17.01.2018 № 67-р. URL: https://zakon.rada.gov.ua/laws/show/67-2018-%D1%80 (дата звернення: 14.05.2026).

4. Семенюк О. А., Кирилащук Т. Г., Січко Т. В. Прикладні аспекти обробки даних в інформаційних системах. Комп'ютерні технології обробки даних. 2021. С. 212–213.

5. Chen C. P. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data / C. P. Chen, C. Y. Zhang // Information Sciences. - 2020. - Vol. 275. - P. 314–347. URL: https://www.researchgate.net/publication/262305146_Data-intensive_applications_challenges_techniques_and_technologies_A_survey_on_Big_Data (дата звернення: 14.05.2026).

6. coverage.py developers. coverage.py Documentation [Електронний ресурс]. - 2024. - URL: https://coverage.readthedocs.io/ (дата звернення: 14.05.2026).

7. Ткачук Н. О., Січко Т. В. Застосування Big Data у бізнесі. Комп'ютерні технології обробки даних. 2023. С. 224–226.

8. Ferrari A. Microsoft Power BI: The Definitive Guide / A. Ferrari, M. Russo. - 2nd ed. - Sebastopol : O'Reilly Media, 2021. - 744 p. URL: https://www.oreilly.com/library/view/definitive-guide-to/9780134865867/ (дата звернення: 14.05.2026).

9. Kimball R. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling / R. Kimball, M. Ross. - 4th ed. - Indianapolis : Wiley, 2023. - 800 p. URL: https://www.oreilly.com/library/view/the-data-warehouse/9781118530801/ (дата звернення: 14.05.2026).

10. Matplotlib Development Team. Matplotlib Documentation [Електронний ресурс]. - 2024. - URL: https://matplotlib.org/stable/contents.html (дата звернення: 14.05.2026).

11. McKinney W. Python for Data Analysis / W. McKinney. - 3rd ed. - Sebastopol : O'Reilly Media, 2022. - 576 p. URL: https://www.oreilly.com/library/view/python-for-data/9781098104023/ (дата звернення: 14.05.2026).

12. Meta Platforms Inc. Prophet: Forecasting at Scale [Електронний ресурс]. - 2023. - URL: https://facebook.github.io/prophet/ (дата звернення: 14.05.2026).

13. Meta Platforms Inc. Prophet GitHub Repository [Електронний ресурс]. - 2023. - URL: https://github.com/facebook/prophet (дата звернення: 14.05.2026).

14. Microsoft Corporation. Power BI Blog [Електронний ресурс]. - 2024. - URL: https://powerbi.microsoft.com/en-us/blog/ (дата звернення: 14.05.2026).

15. Microsoft Corporation. Power BI DAX Function Reference [Електронний ресурс]. - 2024. - URL: https://learn.microsoft.com/en-us/dax/ (дата звернення: 14.05.2026).

16. NumPy Community. NumPy Documentation [Електронний ресурс]. - 2024. - URL: https://numpy.org/doc/stable/ (дата звернення: 14.05.2026).

17. openpyxl contributors. openpyxl Documentation [Електронний ресурс]. - 2024. - URL: https://openpyxl.readthedocs.io/ (дата звернення: 14.05.2026).

18. pandas Documentation [Електронний ресурс]. - 2024. - URL: https://pandas.pydata.org/docs/ (дата звернення: 14.05.2026).

19. PostgreSQL Global Development Group. PostgreSQL Documentation 15 [Електронний ресурс]. - 2024. - URL: https://www.postgresql.org/docs/15/ (дата звернення: 14.05.2026).

20. psycopg2 contributors. psycopg2 Documentation [Електронний ресурс]. - 2024. - URL: https://www.psycopg.org/docs/ (дата звернення: 14.05.2026).

21.pytest team. pytest Documentation [Електронний ресурс]. - 2024. - URL: https://docs.pytest.org/ (дата звернення: 14.05.2026).

22.python-dotenv contributors. python-dotenv Documentation [Електронний ресурс]. - 2024. - URL: https://saurabh-kumar.com/python-dotenv/ (дата звернення: 14.05.2026).

23.scikit-learn developers. scikit-learn Documentation [Електронний ресурс]. - 2024. - URL: https://scikit-learn.org/stable/ (дата звернення: 14.05.2026).

24.Shmueli G. Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python / G. Shmueli, P. C. Bruce, P. Gedeck. - 2nd ed. - Hoboken : Wiley, 2023. - 608 p. URL: https://dokumen.pub/data-mining-for-business-analytics-concepts-techniques-and-applications-in-python.html (дата звернення: 14.05.2026).

25.SQLAlchemy Authors. SQLAlchemy Documentation [Електронний ресурс]. - 2024. - URL: https://docs.sqlalchemy.org/ (дата звернення: 14.05.2026).

26.Січко Т. В. Методи моделювання бізнес-процесів підприємства засобами системного аналізу. Галицький економічний вісник. 2016. № 2(51). С. 190–201. URL: https://elartu.tntu.edu. ua/handle/123456789/20640.

27.Нескородєва Т.В., Федоров Є.Є., Січко Т.В., Нескородєва А.Р. Експертні та рекомендаційні системи: навч. посібник. Вінниця: ДонНУ імені Василя Стуса, 2022, 208 с.

28.\

ДОДАТКИ

ДОДАТОК А

Лістинги коду ETL-конвеєра

А.1. Файл etl/config.py - конфігурація параметрів

```
import os
from dataclasses import dataclass, field
from typing import List

from dotenv import load_dotenv

load_dotenv()

@dataclass
class ETLConfig:
    db_host: str = os.getenv("DB_HOST", "localhost")
    db_port: int = int(os.getenv("DB_PORT", "5432"))
    db_name: str = os.getenv("DB_NAME", "retail_bi")
    db_user: str = os.getenv("DB_USER", "postgres")
    db_password: str = os.getenv("DB_PASSWORD", "postgres")

    dataset_dir: str = os.getenv("DATASET_DIR",
os.path.join(os.path.dirname(__file__), "..", "datasets"))

    products_file: str = os.getenv("PRODUCTS_FILE",
"products.csv")
    customers_file: str = os.getenv("CUSTOMERS_FILE",
"customers.csv")
    stores_file: str = os.getenv("STORES_FILE", "stores.csv")
    sales_file: str = os.getenv("SALES_FILE", "sales.csv")

    log_level: str = os.getenv("LOG_LEVEL", "INFO")
    log_file: str = os.getenv("LOG_FILE", "etl.log")

    chunk_size: int = int(os.getenv("CHUNK_SIZE", "5000"))
    max_retries: int = int(os.getenv("MAX_RETRIES", "3"))

    @property
    def database_url(self) -> str:
        return (
            f"postgresql://{self.db_user}:{self.db_password}"
            f"@{self.db_host}:{self.db_port}/{self.db_name}"
        )

    @property
    def file_paths(self) -> dict:
        return {
```

```
                "products": os.path.join(self.dataset_dir,
self.products_file),
                "customers": os.path.join(self.dataset_dir,
self.customers_file),
                "stores": os.path.join(self.dataset_dir,
self.stores_file),
                "sales": os.path.join(self.dataset_dir,
self.sales_file),
            }

        def validate(self) -> List[str]:
            errors: List[str] = []
            for name, path in self.file_paths.items():
                if not os.path.exists(path):
                    errors.append(f"Missing dataset file:
{path}")
            return errors
```

А.2. Файл etl/transform.py - основна логіка трансформації

```
    from typing import Dict, Tuple

    import pandas as pd
    import numpy as np

    from etl.config import ETLConfig
    from etl.logger import setup_logger

    class DataTransformer:
        def __init__(self, config: ETLConfig):
            self.config = config
            self.logger = setup_logger("transform",
config.log_level, config.log_file)

        def transform_products(self, df: pd.DataFrame) ->
pd.DataFrame:
            self.logger.info("Transforming products data")
            df = df.copy()

            df["product_name"] =
df["product_name"].str.strip().str.title()
            df["category"] =
df["category"].str.strip().str.title()
            df["brand"] = df["brand"].str.strip().str.title()
            df["brand"] =
df["brand"].fillna("Unknown").replace("", "Unknown")

            df["purchase_price"] =
pd.to_numeric(df["purchase_price"], errors="coerce")
            df["sale_price"] = pd.to_numeric(df["sale_price"],
errors="coerce")

            df["purchase_price"] =
df["purchase_price"].clip(lower=0)
```

```
            df["sale_price"] = df["sale_price"].clip(lower=0)

            invalid_prices = df[df["sale_price"] <
df["purchase_price"]]
            if not invalid_prices.empty:
                self.logger.warning(f"Found {len(invalid_prices)}
products with sale_price < purchase_price")
                df.loc[df["sale_price"] < df["purchase_price"],
"sale_price"] = (
                    df.loc[df["sale_price"] <
df["purchase_price"], "purchase_price"] * 1.15
                )

            df = df.drop_duplicates(subset=["product_id"])
            self.logger.info(f"Products transformed: {len(df)}
records")
            return df

        def transform_customers(self, df: pd.DataFrame) ->
pd.DataFrame:
            self.logger.info("Transforming customers data")
            df = df.copy()

            df["first_name"] =
df["first_name"].str.strip().str.title()
            df["last_name"] =
df["last_name"].str.strip().str.title()
            df["last_name"] = df["last_name"].fillna("")
            df["city"] = df["city"].str.strip().str.title()
            df["city"] = df["city"].fillna("Unknown")

            df["age"] = pd.to_numeric(df["age"], errors="coerce")
            median_age = df["age"].median()
            df["age"] = df["age"].fillna(median_age).astype(int)
            df["age"] = df["age"].clip(lower=0, upper=120)

            df["gender"] = df["gender"].str.strip().str.title()
            valid_genders = {"Male", "Female", "Other"}
            df["gender"] = df["gender"].apply(
                lambda x: x if x in valid_genders else "Other"
            )

            df["email"] = df["email"].str.strip().str.lower()
            df["email"] = df["email"].replace("", np.nan)

            df = df.drop_duplicates(subset=["customer_id"])
            self.logger.info(f"Customers transformed: {len(df)}
records")
            return df

        def transform_stores(self, df: pd.DataFrame) ->
pd.DataFrame:
```

```
            self.logger.info("Transforming stores data")
            df = df.copy()

            df["store_name"] = df["store_name"].str.strip()
            df["region"] = df["region"].str.strip().str.title()
            df["city"] = df["city"].str.strip().str.title()
            df["city"] = df["city"].fillna("")

            df = df.drop_duplicates(subset=["store_id"])
            self.logger.info(f"Stores transformed: {len(df)}
records")
            return df

        def transform_sales(self, df: pd.DataFrame) ->
pd.DataFrame:
            self.logger.info("Transforming sales data")
            df = df.copy()

            df["sale_date"] = pd.to_datetime(df["sale_date"],
errors="coerce")
            before_count = len(df)
            df = df.dropna(subset=["sale_date"])
            dropped = before_count - len(df)
            if dropped:
                self.logger.warning(f"Dropped {dropped} sales
with invalid dates")

            df["quantity"] = pd.to_numeric(df["quantity"],
errors="coerce")
            df["quantity"] =
df["quantity"].fillna(1).clip(lower=1).astype(int)

            df["total_amount"] =
pd.to_numeric(df["total_amount"], errors="coerce")
            df["total_amount"] =
df["total_amount"].fillna(0).clip(lower=0)

            df["profit"] = pd.to_numeric(df["profit"],
errors="coerce")
            df["profit"] = df["profit"].fillna(0)

            df["discount"] = pd.to_numeric(df["discount"],
errors="coerce")
            df["discount"] =
df["discount"].fillna(0).clip(lower=0, upper=100)

            df["payment_method"] =
df["payment_method"].str.strip().str.title()
            df["payment_method"] =
df["payment_method"].fillna("Cash")

            df = df.drop_duplicates(subset=["sale_id"])
```

```
            df =
df.sort_values("sale_date").reset_index(drop=True)

            self.logger.info(f"Sales transformed: {len(df)}
records")
            return df

        def transform_all(self, raw_data: Dict[str,
pd.DataFrame]) -> Dict[str, pd.DataFrame]:
            self.logger.info("Starting transformation of all
datasets")
            transformed = {
                "products":
self.transform_products(raw_data["products"]),
                "customers":
self.transform_customers(raw_data["customers"]),
                "stores":
self.transform_stores(raw_data["stores"]),
                "sales": self.transform_sales(raw_data["sales"]),
            }
            self.logger.info("Transformation complete")
            return transformed

        def validate_referential_integrity(
            self, data: Dict[str, pd.DataFrame]
        ) -> Tuple[Dict[str, pd.DataFrame], Dict[str, int]]:
            self.logger.info("Validating referential integrity")
            issues = {}

            sales = data["sales"]
            products = data["products"]["product_id"].unique()
            customers = data["customers"]["customer_id"].unique()
            stores = data["stores"]["store_id"].unique()

            invalid_products =
~sales["product_id"].isin(products)
            if invalid_products.any():
                count = invalid_products.sum()
                self.logger.warning(f"Dropping {count} sales with
invalid product_id")
                sales = sales[~invalid_products]
                issues["invalid_products"] = count

            invalid_customers =
~sales["customer_id"].isin(customers)
            if invalid_customers.any():
                count = invalid_customers.sum()
                self.logger.warning(f"Dropping {count} sales with
invalid customer_id")
                sales = sales[~invalid_customers]
                issues["invalid_customers"] = count
```

```
            invalid_stores = ~sales["store_id"].isin(stores)
            if invalid_stores.any():
                count = invalid_stores.sum()
                self.logger.warning(f"Dropping {count} sales with
invalid store_id")
                sales = sales[~invalid_stores]
                issues["invalid_stores"] = count

            data["sales"] = sales
            self.logger.info(f"Referential integrity check
complete. Issues: {issues}")
            return data, issues
```

А.3. Файл etl/main.py - оркестрація конвеєра

```
#!/usr/bin/env python3
"""
retail-bi-system ETL Pipeline
Main entry point for the Extract-Transform-Load process.
"""

import sys
import time
from pathlib import Path

# Ensure project root is on path
sys.path.insert(0,
str(Path(__file__).resolve().parent.parent))

from etl.config import ETLConfig
from etl.extract import DataExtractor
from etl.transform import DataTransformer
from etl.load import DataLoader
from etl.logger import setup_logger

def run_etl(config_path: str = None) -> int:
    start_time = time.time()
    config = ETLConfig()
    logger = setup_logger("main", config.log_level,
config.log_file)

    logger.info("=" * 60)
    logger.info("RETAIL BI SYSTEM - ETL PIPELINE")
    logger.info("=" * 60)

    validation_errors = config.validate()
    if validation_errors:
        logger.error("Configuration validation failed:")
        for err in validation_errors:
            logger.error(f"  - {err}")
        return 1

    try:
        # Step 1: Extract
```

```
            logger.info("PHASE 1: EXTRACTION")
            extractor = DataExtractor(config)
            raw_data = extractor.extract_all()

            # Step 2: Transform
            logger.info("PHASE 2: TRANSFORMATION")
            transformer = DataTransformer(config)
            transformed_data =
transformer.transform_all(raw_data)

            # Step 3: Validate referential integrity
            logger.info("PHASE 3: REFERENTIAL INTEGRITY CHECK")
            validated_data, issues =
transformer.validate_referential_integrity(transformed_data)
            if issues:
                logger.warning(f"Referential integrity issues
resolved: {issues}")

            # Step 4: Load
            logger.info("PHASE 4: LOADING")
            loader = DataLoader(config)
            verification = loader.load_all(validated_data,
truncate_first=True)

            elapsed = time.time() - start_time
            logger.info("=" * 60)
            logger.info(f"ETL PIPELINE COMPLETED SUCCESSFULLY in
{elapsed:.2f}s")
            logger.info(f"Verification: {verification}")
            logger.info("=" * 60)

            return 0

        except Exception as e:
            logger.error(f"ETL pipeline failed: {e}",
exc_info=True)
            return 1

    if __name__ == "__main__":
        sys.exit(run_etl())
```

ДОДАТОК Б

Лістинги аналітичних модулів Python

Б.1. Файл analytics/forecasting.py - прогнозування продажів (Prophet)

```
#!/usr/bin/env python3
"""
Sales Forecasting Module
Uses Prophet to forecast monthly sales with trend analysis
and accuracy evaluation.
"""

import sys
import warnings
from pathlib import Path

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

sys.path.insert(0,
str(Path(__file__).resolve().parent.parent))

from etl.config import ETLConfig
from etl.logger import setup_logger

class SalesForecaster:
    def __init__(self, config: ETLConfig = None):
        self.config = config or ETLConfig()
        self.logger = setup_logger("forecasting",
self.config.log_level, self.config.log_file)
        self.model = None
        self.forecast = None

    def load_sales_data(self, file_path: str = None) ->
pd.DataFrame:
        path = file_path or self.config.file_paths["sales"]
        self.logger.info(f"Loading sales data from {path}")
        df = pd.read_csv(path, parse_dates=["sale_date"])
        return df

    def prepare_monthly_data(self, df: pd.DataFrame) ->
pd.DataFrame:
        self.logger.info("Aggregating sales to monthly
level")

        df["sale_date"] = pd.to_datetime(df["sale_date"])
        df["month"] =
df["sale_date"].dt.to_period("M").dt.to_timestamp()
```

```
            monthly = (
                df.groupby("month")
                .agg(revenue=("total_amount", "sum"),
transactions=("sale_id", "nunique"), quantity=("quantity", "sum"))
                .reset_index()
            )

            monthly =
monthly.sort_values("month").rename(columns={"month": "ds",
"revenue": "y"})

            self.logger.info(f"Prepared {len(monthly)} monthly
data points")
            return monthly

        def train_prophet(self, monthly_df: pd.DataFrame):
            try:
                from prophet import Prophet
            except ImportError:
                self.logger.error(
                    "Prophet not installed. Run: pip install
prophet"
                )
                raise

            self.logger.info("Training Prophet forecasting
model")

            model = Prophet(
                yearly_seasonality=True,
                weekly_seasonality=False,
                daily_seasonality=False,
                seasonality_mode="multiplicative",
                changepoint_prior_scale=0.05,
                seasonality_prior_scale=10.0,
                interval_width=0.95,
            )

            model.add_seasonality(name="monthly", period=30.5,
fourier_order=5)
            model.fit(monthly_df)

            self.model = model
            self.logger.info("Prophet model training complete")
            return model

        def forecast_future(self, periods: int = 12) ->
pd.DataFrame:
            if self.model is None:
                raise ValueError("Model not trained. Call
train_prophet first.")
```

```
            self.logger.info(f"Generating forecast for {periods}
months")
            future =
self.model.make_future_dataframe(periods=periods, freq="ME")
            forecast = self.model.predict(future)
            self.forecast = forecast
            return forecast

        def evaluate_accuracy(self, monthly_df: pd.DataFrame) ->
dict:
            if self.forecast is None:
                raise ValueError("No forecast available. Run
forecast_future first.")

            historical = monthly_df.copy()
            historical["predicted"] = self.forecast.loc[:
len(historical) - 1, "yhat"].values
            historical = historical.dropna()

            mae = np.abs(historical["y"] -
historical["predicted"]).mean()
            mape = (np.abs((historical["y"] -
historical["predicted"]) / historical["y"]) * 100).mean()
            rmse = np.sqrt(((historical["y"] -
historical["predicted"]) ** 2).mean())

            metrics = {
                "mae": round(mae, 2),
                "mape": round(mape, 2),
                "rmse": round(rmse, 2),
                "accuracy": round(100 - mape, 2),
            }

            self.logger.info(f"Forecast accuracy: {metrics}")
            return metrics

        def plot_forecast(
            self, monthly_df: pd.DataFrame, output_path: str =
"docs/forecast.png"
        ):
            if self.model is None or self.forecast is None:
                raise ValueError("Run train_prophet and
forecast_future first.")

            fig, axes = plt.subplots(2, 2, figsize=(16, 10))

            ax1 = axes[0, 0]
            ax1.plot(monthly_df["ds"], monthly_df["y"], "b-",
label="Actual", linewidth=2)
            ax1.plot(
                self.forecast["ds"],
```

```
                self.forecast["yhat"],
                "r--",
                label="Forecast",
                linewidth=2,
                alpha=0.8,
            )
            ax1.fill_between(
                self.forecast["ds"],
                self.forecast["yhat_lower"],
                self.forecast["yhat_upper"],
                color="red",
                alpha=0.15,
                label="95% Confidence Interval",
            )
            ax1.axvline(
                x=monthly_df["ds"].max(),
                color="gray",
                linestyle=":",
                alpha=0.7,
                label="Forecast Start",
            )
            ax1.set_title("Sales Forecast (Prophet)",
fontsize=14, fontweight="bold")
            ax1.set_xlabel("Date")
            ax1.set_ylabel("Revenue")
            ax1.legend()
            ax1.grid(True, alpha=0.3)

            ax2 = axes[0, 1]
            historical_pred = self.forecast.loc[: len(monthly_df)
- 1]
            ax2.scatter(monthly_df["y"], historical_pred["yhat"],
alpha=0.6, color="#3498db")
            max_val = max(monthly_df["y"].max(),
historical_pred["yhat"].max())
            ax2.plot([0, max_val], [0, max_val], "r--",
alpha=0.5)
            ax2.set_title("Actual vs Predicted", fontsize=14,
fontweight="bold")
            ax2.set_xlabel("Actual Revenue")
            ax2.set_ylabel("Predicted Revenue")
            ax2.grid(True, alpha=0.3)

            ax3 = axes[1, 0]
            ax3.plot(
                self.forecast["ds"],
                self.forecast["trend"],
                "g-",
                label="Trend",
                linewidth=2,
            )
```

```
            ax3.set_title("Forecast Trend Component",
fontsize=14, fontweight="bold")
            ax3.set_xlabel("Date")
            ax3.set_ylabel("Trend Value")
            ax3.grid(True, alpha=0.3)

            ax4 = axes[1, 1]
            monthly_comp =
self.forecast.set_index("ds")["yearly"].to_frame()
            monthly_comp =
monthly_comp.groupby(monthly_comp.index.month).mean()
            ax4.bar(
                monthly_comp.index,
                monthly_comp["yearly"],
                color="#2ecc71",
                edgecolor="white",
            )
            ax4.set_title("Seasonal Pattern (Monthly)",
fontsize=14, fontweight="bold")
            ax4.set_xlabel("Month")
            ax4.set_ylabel("Seasonal Effect")
            ax4.set_xticks(range(1, 13))
            ax4.set_xticklabels(
                ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
                 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
            )
            ax4.grid(True, alpha=0.3)

            plt.tight_layout()
            Path(output_path).parent.mkdir(parents=True,
exist_ok=True)
            plt.savefig(output_path, dpi=150,
bbox_inches="tight")
            self.logger.info(f"Forecast plot saved to
{output_path}")

        def export_forecast(self, output_path: str =
"datasets/forecast_results.csv"):
            if self.forecast is None:
                raise ValueError("No forecast available.")

            export_df = self.forecast[["ds", "yhat",
"yhat_lower", "yhat_upper", "trend"]].copy()
            export_df.columns = ["date", "predicted_revenue",
"lower_bound", "upper_bound", "trend"]
            export_df["date"] =
export_df["date"].dt.strftime("%Y-%m-%d")

            full_path = Path(__file__).resolve().parent.parent /
output_path
            full_path.parent.mkdir(parents=True, exist_ok=True)
            export_df.to_csv(full_path, index=False)
```

```
            self.logger.info(f"Forecast exported to {full_path}")
            return full_path

    def main():
        forecaster = SalesForecaster()

        sales_df = forecaster.load_sales_data()
        monthly_df = forecaster.prepare_monthly_data(sales_df)

        forecaster.train_prophet(monthly_df)
        forecast = forecaster.forecast_future(periods=12)

        metrics = forecaster.evaluate_accuracy(monthly_df)
        forecaster.plot_forecast(monthly_df)
        forecaster.export_forecast()

        print("\nSales Forecasting Results:")
        print("=" * 60)
        print(f"Model: Prophet (Multiplicative Seasonality)")
        print(f"Training data points: {len(monthly_df)} months")
        print(f"Forecast horizon: 12 months")
        print(f"\nAccuracy Metrics:")
        for key, value in metrics.items():
            unit = "%" if key in ("mape", "accuracy") else ""
            print(f"  {key.upper()}: {value}{unit}")

        print(f"\nNext 6 months forecast:")
        future_forecast = forecast[forecast["ds"] >
monthly_df["ds"].max()].head(6)
        for _, row in future_forecast.iterrows():
            print(f"  {row['ds'].strftime('%Y-%m')}: "
                  f"${row['yhat']:,.2f} "
                  f"(${row['yhat_lower']:,.2f} -
${row['yhat_upper']:,.2f})")

        return 0

    if __name__ == "__main__":
        sys.exit(main())
```

Б.2. Файл analytics/abc_analysis.py - ABC-класифікація товарів

```
    #!/usr/bin/env python3
    """
    ABC Analysis Module
    Classifies products into A, B, C categories based on revenue
contribution.
    A: top 80% revenue
    B: next 15% revenue
    C: bottom 5% revenue
    """

    import sys
    from pathlib import Path
```

```
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    sys.path.insert(0,
str(Path(__file__).resolve().parent.parent))

    from etl.config import ETLConfig
    from etl.logger import setup_logger

    class ABCAnalyzer:
        def __init__(self, config: ETLConfig = None):
            self.config = config or ETLConfig()
            self.logger = setup_logger("abc_analysis",
self.config.log_level, self.config.log_file)

        def load_sales_data(self, file_path: str = None) ->
pd.DataFrame:
            path = file_path or self.config.file_paths["sales"]
            self.logger.info(f"Loading sales data from {path}")
            df = pd.read_csv(path, parse_dates=["sale_date"])
            return df

        def load_products_data(self, file_path: str = None) ->
pd.DataFrame:
            path = file_path or
self.config.file_paths["products"]
            self.logger.info(f"Loading products data from
{path}")
            df = pd.read_csv(path)
            return df

        def analyze(self, sales_df: pd.DataFrame, products_df:
pd.DataFrame) -> pd.DataFrame:
            self.logger.info("Starting ABC analysis")

            merged = sales_df.merge(products_df, on="product_id",
how="left")

            product_revenue = (
                merged.groupby(["product_id", "product_name",
"category", "brand"])
                .agg(total_revenue=("total_amount", "sum"),
total_profit=("profit", "sum"))
                .reset_index()
            )

            product_revenue =
product_revenue.sort_values("total_revenue",
ascending=False).reset_index(drop=True)
```

```
            total_revenue_all =
product_revenue["total_revenue"].sum()
            product_revenue["revenue_pct"] = (
                product_revenue["total_revenue"] /
total_revenue_all * 100
            )
            product_revenue["cumulative_revenue_pct"] =
product_revenue["revenue_pct"].cumsum()

            conditions = [
                product_revenue["cumulative_revenue_pct"] <= 80,
                product_revenue["cumulative_revenue_pct"] <= 95,
            ]
            choices = ["A", "B"]
            product_revenue["abc_category"] =
np.select(conditions, choices, default="C")

            self.logger.info(
                f"ABC Classification: "
                f"A={len(product_revenue[product_revenue['abc_cat
egory']=='A'])}, "
                f"B={len(product_revenue[product_revenue['abc_cat
egory']=='B'])}, "
                f"C={len(product_revenue[product_revenue['abc_cat
egory']=='C'])}"
            )

            return product_revenue

        def plot_abc_curve(self, df: pd.DataFrame, output_path:
str = "docs/abc_analysis.png"):
            fig, axes = plt.subplots(1, 2, figsize=(14, 6))

            ax1 = axes[0]
            colors = {"A": "#2ecc71", "B": "#f39c12", "C":
"#e74c3c"}
            bar_colors = [colors.get(cat, "#95a5a6") for cat in
df["abc_category"]]
            ax1.bar(range(len(df)), df["total_revenue"],
color=bar_colors, alpha=0.8)
            ax1.set_title("Product Revenue by ABC Category",
fontsize=14, fontweight="bold")
            ax1.set_xlabel("Products (sorted by revenue)")
            ax1.set_ylabel("Total Revenue")
            ax1.axhline(y=df["total_revenue"].max() * 0.8,
color="green", linestyle="--", alpha=0.5)

            from matplotlib.patches import Patch
            legend_elements = [
                Patch(facecolor="#2ecc71", label="A - Top 80%"),
                Patch(facecolor="#f39c12", label="B - Next 15%"),
```

```
                Patch(facecolor="#e74c3c", label="C - Bottom
5%"),
            ]
            ax1.legend(handles=legend_elements)

            ax2 = axes[1]
            abc_counts =
df["abc_category"].value_counts().sort_index()
            wedges, texts, autotexts = ax2.pie(
                abc_counts.values,
                labels=abc_counts.index,
                autopct="%1.1f%%",
                colors=["#2ecc71", "#f39c12", "#e74c3c"],
                startangle=90,
                explode=(0.05, 0.05, 0.05),
            )
            ax2.set_title("ABC Category Distribution (Product
Count)", fontsize=14, fontweight="bold")

            plt.tight_layout()
            Path(output_path).parent.mkdir(parents=True,
exist_ok=True)
            plt.savefig(output_path, dpi=150,
bbox_inches="tight")
            self.logger.info(f"ABC analysis plot saved to
{output_path}")

        def export_abc_data(self, df: pd.DataFrame, output_path:
str = "datasets/abc_classification.csv"):
            full_path = Path(__file__).resolve().parent.parent /
output_path
            full_path.parent.mkdir(parents=True, exist_ok=True)
            df.to_csv(full_path, index=False)
            self.logger.info(f"ABC classification exported to
{full_path}")

    def main():
        analyzer = ABCAnalyzer()
        sales = analyzer.load_sales_data()
        products = analyzer.load_products_data()
        abc_results = analyzer.analyze(sales, products)

        print("\nABC Analysis Results:")
        print("=" * 60)
        for category in ["A", "B", "C"]:
            cat_df = abc_results[abc_results["abc_category"] ==
category]
            print(f"\nCategory {category}: {len(cat_df)}
products")
            print(f"  Total Revenue:
{cat_df['total_revenue'].sum():,.2f}")
```

```
            print(f"  Top product:
{cat_df.iloc[0]['product_name']}
({cat_df.iloc[0]['total_revenue']:,.2f})")

        analyzer.plot_abc_curve(abc_results)
        analyzer.export_abc_data(abc_results)
        return 0

    if __name__ == "__main__":
        sys.exit(main())
```

Б.3. Файл analytics/customer_analysis.py - сегментація клієнтів

```
    #!/usr/bin/env python3
    """
    Customer Analysis Module
    Analyzes customer purchase behavior, segmentation, and value
metrics.
    """

    import sys
    from pathlib import Path

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    sys.path.insert(0,
str(Path(__file__).resolve().parent.parent))

    from etl.config import ETLConfig
    from etl.logger import setup_logger

    class CustomerAnalyzer:
        def __init__(self, config: ETLConfig = None):
            self.config = config or ETLConfig()
            self.logger = setup_logger("customer_analysis",
self.config.log_level, self.config.log_file)

        def load_data(self) -> tuple:
            sales = pd.read_csv(self.config.file_paths["sales"],
parse_dates=["sale_date"])
            customers =
pd.read_csv(self.config.file_paths["customers"])
            self.logger.info(f"Loaded {len(sales)} sales and
{len(customers)} customers")
            return sales, customers

        def analyze(self, sales_df: pd.DataFrame, customers_df:
pd.DataFrame) -> pd.DataFrame:
            self.logger.info("Starting customer analysis")

            merged = sales_df.merge(customers_df,
on="customer_id", how="left")
```

```
            customer_metrics = (
                merged.groupby("customer_id")
                .agg(
                    first_name=("first_name", "first"),
                    last_name=("last_name", "first"),
                    city=("city", "first"),
                    age=("age", "first"),
                    gender=("gender", "first"),
                    purchase_count=("sale_id", "nunique"),
                    total_revenue=("total_amount", "sum"),
                    total_profit=("profit", "sum"),
                    avg_order_value=("total_amount", "mean"),
                    total_quantity=("quantity", "sum"),
                    first_purchase=("sale_date", "min"),
                    last_purchase=("sale_date", "max"),
                )
                .reset_index()
            )

            customer_metrics["customer_name"] = (
                customer_metrics["first_name"] + " " +
customer_metrics["last_name"]
            )

            analysis_period_days = (
                customer_metrics["last_purchase"].max() -
customer_metrics["first_purchase"].min()
            ).days

            customer_metrics["purchase_frequency"] = (
                customer_metrics["purchase_count"] /
max(analysis_period_days, 1) * 30
            )

            revenue_bins = [0, 400, 700, 1000, 1500,
float("inf")]
            revenue_labels = ["Standard", "Bronze", "Silver",
"Gold", "Platinum"]
            customer_metrics["tier"] = pd.cut(
                customer_metrics["total_revenue"],
bins=revenue_bins, labels=revenue_labels, right=True
            )

            freq_bins = [0, 1, 3, 6, 10, float("inf")]
            freq_labels = ["At Risk", "Occasional", "Regular",
"Frequent", "VIP"]
            customer_metrics["engagement"] = pd.cut(
                customer_metrics["purchase_count"],
bins=freq_bins, labels=freq_labels, right=True
            )
```

```
            self.logger.info(
                f"Customer tiers: "
                f"{customer_metrics['tier'].value_counts().to_dic
t()}"
            )

            return customer_metrics

        def plot_customer_segmentation(self, df: pd.DataFrame,
output_path: str = "docs/customer_segmentation.png"):
            fig, axes = plt.subplots(2, 2, figsize=(14, 12))

            tier_colors = {
                "Platinum": "#8e44ad",
                "Gold": "#f1c40f",
                "Silver": "#95a5a6",
                "Bronze": "#e67e22",
                "Standard": "#7f8c8d",
            }

            ax1 = axes[0, 0]
            tier_counts = df["tier"].value_counts()
            bars = ax1.bar(
                tier_counts.index,
                tier_counts.values,
                color=[tier_colors.get(t, "#95a5a6") for t in
tier_counts.index],
                edgecolor="white",
                linewidth=1.5,
            )
            ax1.set_title("Customer Tier Distribution",
fontsize=13, fontweight="bold")
            ax1.set_xlabel("Tier")
            ax1.set_ylabel("Customer Count")
            for bar, val in zip(bars, tier_counts.values):
                ax1.text(bar.get_x() + bar.get_width() / 2,
bar.get_height() + 0.5,
                         str(val), ha="center",
fontweight="bold")

            ax2 = axes[0, 1]
            engagement_counts = df["engagement"].value_counts()
            colors = ["#e74c3c", "#e67e22", "#f1c40f", "#2ecc71",
"#3498db"]
            wedges, texts, autotexts = ax2.pie(
                engagement_counts.values,
                labels=engagement_counts.index,
                autopct="%1.1f%%",
                colors=colors[: len(engagement_counts)],
                startangle=90,
                explode=[0.02] * len(engagement_counts),
            )
```

```
            ax2.set_title("Customer Engagement Distribution",
fontsize=13, fontweight="bold")

            ax3 = axes[1, 0]
            revenue_by_tier =
df.groupby("tier")["total_revenue"].sum().sort_values(ascending=Fa
lse)
            ax3.bar(
                revenue_by_tier.index,
                revenue_by_tier.values / 1000,
                color=[tier_colors.get(t, "#95a5a6") for t in
revenue_by_tier.index],
                edgecolor="white",
                linewidth=1.5,
            )
            ax3.set_title("Revenue by Customer Tier",
fontsize=13, fontweight="bold")
            ax3.set_xlabel("Tier")
            ax3.set_ylabel("Total Revenue (thousands)")

            ax4 = axes[1, 1]
            ax4.hist(df["purchase_count"], bins=30,
color="#3498db", edgecolor="white", alpha=0.8)
            ax4.set_title("Purchase Frequency Distribution",
fontsize=13, fontweight="bold")
            ax4.set_xlabel("Number of Purchases")
            ax4.set_ylabel("Customer Count")

            plt.tight_layout()
            Path(output_path).parent.mkdir(parents=True,
exist_ok=True)
            plt.savefig(output_path, dpi=150,
bbox_inches="tight")
            self.logger.info(f"Customer segmentation plot saved
to {output_path}")

        def export_results(self, df: pd.DataFrame, output_path:
str = "datasets/customer_metrics.csv"):
            full_path = Path(__file__).resolve().parent.parent /
output_path
            full_path.parent.mkdir(parents=True, exist_ok=True)
            df.to_csv(full_path, index=False)
            self.logger.info(f"Customer metrics exported to
{full_path}")

    def main():
        analyzer = CustomerAnalyzer()
        sales, customers = analyzer.load_data()
        metrics = analyzer.analyze(sales, customers)

        print("\nCustomer Analysis Results:")
        print("=" * 60)
```

```
        print(f"Total customers analyzed: {len(metrics)}")
        print(f"Average purchases per customer:
{metrics['purchase_count'].mean():.1f}")
        print(f"Average revenue per customer:
{metrics['total_revenue'].mean():.2f}")
        print(f"Average order value:
{metrics['avg_order_value'].mean():.2f}")
        print(f"\nCustomer Tier Breakdown:")
        for tier, count in
metrics["tier"].value_counts().sort_index().items():
            print(f"  {tier}: {count} customers")

        analyzer.plot_customer_segmentation(metrics)
        analyzer.export_results(metrics)
        return 0

    if __name__ == "__main__":
        sys.exit(main())
```

ДОДАТОК В

Лістинги SQL-скриптів та DAX-мір

В.1. Файл database/schema.sql - створення таблиць (фрагмент)

```
CREATE TABLE products (
    product_id      SERIAL          PRIMARY KEY,
    product_name    VARCHAR(255)    NOT NULL,
    category        VARCHAR(100)    NOT NULL,
    brand           VARCHAR(100)    NOT NULL DEFAULT 'Unknown',
        purchase_price    NUMERIC(10,   2)    NOT   NULL   CHECK
(purchase_price >= 0),
    sale_price      NUMERIC(10, 2)  NOT NULL CHECK (sale_price
>= 0),
        created_at          TIMESTAMP           NOT  NULL  DEFAULT
CURRENT_TIMESTAMP,
        updated_at          TIMESTAMP           NOT  NULL  DEFAULT
CURRENT_TIMESTAMP
);

-- Indexes for products
CREATE INDEX idx_products_category ON products(category);
CREATE INDEX idx_products_brand ON products(brand);

--
==========================================================
==========
-- TABLE: customers
-- Description: Customer demographics and information
--
==========================================================
==========
CREATE TABLE customers (
    customer_id     SERIAL          PRIMARY KEY,
    first_name      VARCHAR(100)    NOT NULL,
    last_name       VARCHAR(100)    NOT NULL DEFAULT '',
    city            VARCHAR(100)    NOT NULL DEFAULT 'Unknown',
    age              INTEGER          NOT NULL CHECK (age >= 0
AND age <= 120),
     gender          VARCHAR(20)      NOT NULL DEFAULT 'Other'
CHECK (gender IN ('Male', 'Female', 'Other')),
    email           VARCHAR(255)    UNIQUE,
      registration_date  DATE              NOT  NULL  DEFAULT
CURRENT_DATE,
      created_at          TIMESTAMP           NOT  NULL  DEFAULT
CURRENT_TIMESTAMP
);

-- Indexes for customers
CREATE INDEX idx_customers_city ON customers(city);
CREATE INDEX idx_customers_age ON customers(age);
```

```
CREATE INDEX idx_customers_gender ON customers(gender);
```

Перелік DAX-мір Power BI

```
1. Total Revenue = SUM('sales'[total_amount])
2. Total Profit = SUM('sales'[profit])
3. Profit Margin % = DIVIDE([Total Profit], [Total Revenue], 0)
* 100
4. Total Transactions = COUNTROWS('sales')
5. Units Sold = SUM('sales'[quantity])
6. Average Order Value = DIVIDE([Total Revenue], [Total
Transactions], 0)
7. Prev Month Revenue = CALCULATE([Total Revenue],
PREVIOUSMONTH(DateTable[Date]))
8. MoM Growth % = DIVIDE([Total Revenue] - [Prev Month Revenue],
[Prev Month Revenue], 0) * 100
9. YoY Revenue = CALCULATE([Total Revenue],
SAMEPERIODLASTYEAR(DateTable[Date]))
10. YoY Growth % = DIVIDE([Total Revenue] - [YoY Revenue], [YoY
Revenue], 0) * 100
11. Running Total Revenue = CALCULATE([Total Revenue],
DATESYTD(DateTable[Date]))
12. Revenue per Customer = DIVIDE([Total Revenue],
DISTINCTCOUNT('sales'[customer_id]), 0)
```

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
NOMEN EST OMEN
ALMA MATER
ІМ'Я ЗОБОВ'ЯЗУЄ