НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Розробка інформаційної системи для оптимального формування раціону домашніх котів з урахуванням вартісних обмежень та нормативного нутрієнтного забезпечення»»

Виконала:
студентка IV курсу, групи КА-11
Степаненко Анастасія Сергіївна _____

Керівник:
Доцент, к. ф.-м. н., Яковлева Алла Петрівна _____

Консультант з економічного розділу:
доц., к. е. н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
к. ф.-м. н., Статкевич Віталій Михайлович _____

Рецензент:
к. ф.-м. н., доц. каф. матаналізу та теорії ймовірностей,
Ільєнко Андрій Борисович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2025 р.

ЗАВДАННЯ

на дипломну роботу студентці

Степаненко Анастасії Сергіївні

1. Тема роботи «Розробка інформаційної системи для оптимального формування раціону домашніх котів з урахуванням вартісних обмежень та нормативного нутрієнтного забезпечення», керівник роботи Яковлева Алла Петрівна, кандидат фізико-математичних наук, затверджені наказом по університету від 26.05.2025 р. №1759-с

2. Термін подання студентом роботи

3. Вихідні дані до роботи: зібрані дані поїздок.

4. Зміст роботи: дослідження предметної області, теоретичні основи до створення додатку, програмна реалізація та аналіз результатів, функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): зображення тестових результатів оптимізації, візуалізації даних та користувацького інтерфейсу, реалізація модулів фільтрації та зображення початкових даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доц., к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи (БДР)	Термін виконання етапів роботи	Примітка
1	Затвердження теми БДР	13.04.2025-20.05.2025	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	13.04.2025-20.05.2025	виконано
3	Ознайомлення з ДСТУ 3008:2015	13.04.2025-20.05.2025	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	20.04.2025-07.05.2025	виконано
5	Завершення роботи над першим варіантом частини БДР	30.04.2025-14.05.2025	виконано
6	Проведення роботи над експериментальною частиною БДР	30.04.2025-14.05.2025	виконано
7	Проведення роботи над програмним продуктом	30.04.2025-26.05.2025	виконано
8	Оформлення БДР та аналіз отриманих результатів	10.04.2025-03.06.2025	виконано

Студентка Анастасія СТЕПАНЕНКО

Керівник Алла ЯКОВЛЕВА

РЕФЕРАТ

Дипломна робота: 106 с., 24 рис., 9 табл., 2 дод., 23 джерела.

VETERINARY DATA ANALYZER, DASH, PYTHON, PET NUTRITION, OPTIMIZATION, T-SNE, UMAP, ROBUST

Об’єкт дослідження – автоматизація формування збалансованого раціону домашніх котів з урахуванням вартості кормів і нормативів FEDIAF.

Предмет дослідження – застосування Dash для імпорту та обробки CSV/Excel-даних, t-SNE і UMAP для візуалізації нутрієнтних профілів та три режими оптимізації (лінійний, стохастичний ±20 % і робастний) для мінімізації собівартості.

Мета роботи – реалізувати на Python інтерактивну систему, яка фільтрує аномально дорогі продукти, обчислює показники TotalVitPerEuro і CheapCount, класифікує їх як Super-bomb, Bomb чи Inefficient і формує оптимальний набір порцій із точним покриттям мінімальних і максимальних норм.

Методи дослідження – PCA, t-SNE, UMAP, лінійне програмування на PuLP, стохастична та інтервальна робастна оптимізація, інтерактивна візуалізація Plotly.

Актуальність полягає в необхідності поєднати сучасні статистичні й оптимізаційні підходи для економічного й ефективного ветеринарного ризик-менеджменту.

Результат – веб-додаток Veterinary Data Analyzer з модулями імпорту, фільтрації, класифікації, кластеризації, трьома підходами оптимізації та звітами з розподілом маси, вартості й покриттям норм.

Подальший розвиток – розширити нормативну базу, інтегрувати динамічний парсер цін та протестувати альтернативні солвери .

Технології – Python, Dash, pandas, Plotly, scikit-learn, umap-learn, PuLP, SQLite.

ABSTRACT

Diploma thesis: 106 p., 24 fig., 9 tabl., 2 app., 23 references.

VETERINARY DATA ANALYZER, DASH, PYTHON, PET NUTRITION, OPTIMIZATION, T-SNE, UMAP, ROBUST

Research object – the automation of formulating a balanced domestic-cat diet taking into account feed cost and FEDIAF standards.

Research subject – the use of the Dash web framework for importing and preprocessing CSV/Excel data, t-SNE and UMAP for visualizing nutrient profiles, and three optimization modes (linear, stochastic ± 20 %, and robust interval) to minimize diet cost.

Objective – to implement in Python an interactive system that filters out abnormally expensive products, computes the TotalVitPerEuro and CheapCount metrics, classifies items as Super-bomb, Bomb, or Inefficient, and generates an optimal set of servings precisely meeting minimum and maximum nutrient requirements.

Methods – PCA; nonlinear dimensionality reduction via t-SNE and UMAP; linear programming with PuLP; stochastic and interval robust optimization; and interactive Plotly visualization.

Relevance – integrating modern statistical and optimization techniques is essential for cost-effective and reliable veterinary risk management.

Results – the Veterinary Data Analyzer web app, featuring modules for data import, filtering, classification, clustering, three optimization strategies, and automated reports on mass distribution, cost share, and nutrient coverage.

Further development – expand the normative database, integrate dynamic price parsing, and evaluate alternative solvers.

Technologies – Python, Dash, pandas, Plotly, scikit-learn, umap-learn, PuLP, SQLite.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FEDIAF – European Pet Food Industry Federation

NRC – National Research Council

MER— добова потреба в метаболізованій енергії (ккал)

BW — маса тіла у кілограмах

CP (англ. DM — Dry Matter) — суха речовина

t-SNE — t-Stochastic Neighbor Embedding

UMAP — Uniform Manifold Approximation and Projection

ВСТУП

Сучасні дослідження у сфері ветеринарної дієтології підтверджують зростаючу потребу в персоналізованих, економічно обґрунтованих та науково вивірених підходах для формування збалансованого раціону домашніх тварин. Особливу актуальність має оптимізація дієти для котів з урахуванням їх потреб у поживних речовинах та вартості кормових інгредієнтів. З урахуванням наявних допустимих похибок, зокрема, коливання поживних характеристик продуктів у межах до 20%, виникає потреба у створенні інформаційної системи, здатної враховувати стохастичність даних та забезпечувати стабільне рішення.

Дипломна робота присвячена розробці та реалізації інформаційної системи, що ґрунтується на методах стохастичного лінійного програмування та робастної оптимізації. Таке поєднання дозволяє не лише зменшити загальну вартість раціону, але й забезпечити відповідність дієтичним нормам у реальному світі.

Необхідність створення такої системи зумовлена практичними труднощами, з якими стикаються власники домашніх тварин, ветеринарні спеціалісти та виробники кормів при забезпеченні збалансованого харчування. Аналіз літературних джерел, зокрема рекомендацій FEDIAF та NRC, підтверджує, що більшість існуючих підходів орієнтовані або на максимальне наближення до нормативів, або на мінімізацію вартості без урахування можливих відхилень.

Новизна роботи полягає в поетапному підході до оптимізації з урахуванням пріоритетності груп нутрієнтів, побудові адаптивної моделі з використанням Python та інтеграції інструментів лінійного програмування з інтерфейсом користувача для подальшого практичного застосування.

Результати дипломної роботи можуть бути використані у ветеринарній практиці, в автоматизованих системах підтримки прийняття рішень для

власників тварин, а також у сфері виробництва кормів як допоміжний інструмент для раціонального формування рецептур.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Біологічні та фізіологічні потреби котів

Кішка — облігатний хижак, еволюційно пристосований до раціону з дрібної тваринної здобичі; ця спеціалізація зумовлює характерні метаболічні й анатомо-фізіологічні риси, що задають якісні та кількісні параметри повноцінного харчування [1, 2]. Енергетична потреба дорослої активної тварини описується рівнянням $MER = 100 \cdot BW^{0.67}$, тоді як у стерилізованих або переважно домашніх котів вона знижується приблизно до $MER = 75 \cdot BW^{0.67}$. Природна здобич містить 50–60 % перетравного білка в сухій речовині (СР), тому мінімальний рівень протеїну у промислових кормах має становити щонайменше 28–33 г/100 г СР, залежно від енергетичної щільності раціону [2, 3].

Через високу активність печінкових трансаміназ кішка не може суттєво знижувати швидкість білкового катаболізму, що обумовлює підвищені вимоги до таурину, аргініну та інших незамінних амінокислот [1]. Активність кишкових амілази та сахарози у котів знижена; крохмаль у природному раціоні майже відсутній. Регуляція глюкокінази не залежить від надходження вуглеводів, тому частка крохмалю в сухих кормах обмежується його перетравністю, аби запобігти постпрандіальній гіперглікемії та осмотичній діареї [2]. Через відсутність певних ферментів кішки мають специфічні дієтичні потреби:

1) вітамін А потрібен у готовій формі ретинолу (β-каротин не засвоюється) [2];
2) ніацин (B_3) повинен надходити з корму, мінімум 1,13 мг/г СР, через швидкий розпад триптофану [2];

3) вітамін D синтезується шкірою в обмеженій кількості й має надходити з їжею [3];
4) оптимальне співвідношення Ca:P ≈ 1:1–1,5:1, оскільки дефіцит кальцію чи надлишок фосфору призводять до остеодистрофії [3];
5) рекомендований мінімум магнію становить 20 мг/100 ккал ME; його надлишок може сприяти струвітоліту, тому FEDIAF підвищує мінімум з огляду на іонний склад сечі [3].

Природна «високо м'ясна» дієта містить понад 70 % вологи, тоді як сухі корми — ≈ 8–10 %. Слабко виражене відчуття спраги та висока концентраційна здатність нирок підвищують ризик уролітіазу, тому рекомендовано використання вологих раціонів або стимуляцію споживання води при годуванні сухими кормами [4]. Таким чином, специфічні біологічні особливості котів — високий протеїновий запит, потреба в окремих амінокислотах, обмежений синтез деяких вітамінів і жирних кислот, чутливість до зневоднення — обумовлюють жорсткі вимоги до збалансованості раціону. Ці чинники покладено в основу сучасних нормативів FEDIAF і NRC, що надалі використовуються як обмеження у математичних моделях оптимізації складу кормів [2, 3].

1.2 Нормативні бази (FEDIAF, NRC)

Сучасна практика клінічної дієтології спирається на кілька авторитетних нормативних джерел, що задають мінімальні, оптимальні та безпечні рівні поживних речовин для котів. У межах цієї роботи було обрано два найвпливовіші документи — європейські Nutritional Guidelines федерації виробників кормів FEDIAF [3] та монографію Nutrient Requirements of Dogs and Cats Національної академії наук США (NRC) [2]. Кожен із них

послуговується специфічною методологією розрахунку норм і має власні переваги та обмеження.

FEDIAF раз на кілька років оновлює рекомендації (остання редакція — 2024 р.) [3]. Мінімальні та максимальні рівні макро- і мікронутрієнтів подано одразу у трьох масштабах: на 100 г сухої речовини, на 1 000 ккал метаболізованої енергії та на $kg^{0.67}$маси тіла. Документ також упроваджує поправковий коефіцієнт для тварин із меншим енергоспоживанням (< 100 ккал/$kg^{0.67}$), що запобігає «прихованому» дефіциту й робить ці норми особливо доречними для кастрованих і малорухливих домашніх котів.

Монографія NRC (2006) має фундаментально-науковий характер і ґрунтується на контрольованих експериментах та факторному аналізі потреб [2]. Вона подає «мінімальну потребу», «рекомендоване споживання» та «безпечну верхню межу» у грамах на кілограм маси тіла або у % сухої речовини. На відміну від FEDIAF, NRC не коригує мікронутрієнти за фактичним енергоспоживанням: базове рівняння $MER = 100$ ккал/$\cdot kg^{0.67}$ призначене для не кастрованих активних котів, а енергетична щільність корму залишається поза межами моделі.

1.3 Сучасні підходи до формування раціону

Еволюція клінічної дієтології котів від загальних «середніх» норм перейшла до індивідуалізованих раціонів, у яких враховуються вік, фізіологічний статус, поведінкові особливості, коморбідні патології та навіть порода тварини. Центральною парадигмою такого підходу стала концепція «п’яти кроків нутрітивного процесу» (скринінг → розгорнута оцінка → план → моніторинг → корекція), докладно описана у спільних рекомендаціях AAHA та WSAVA 2021 р. [5]. Модель WSAVA/AAHA передбачає

багатофакторний збір даних (історія годівлі, BCS, MCS, лабораторні маркери, поведінкові та середовищні чинники) з подальшим зіставленням ризику дефіциту чи надлишку нутрієнтів, що мінімізує харчову ятрогенію [4, 5].

Новітні керівництва з клінічної дієтології наголошують на пріоритеті біодоступності та функціональної ролі окремих фракцій: високоякісних тваринних білків, DHA/EPA як довголанцюгових n-3 ПНЖК, контрольованого рівня крохмалю з огляду на низьку активність дисахараз у котів [1].

Індустрія дедалі частіше випускає лінійки «модульних» кормів, орієнтованих на конкретні клінічні потреби (контроль маси, ШКТ-підтримка, уролітіаз тощо). Алгоритм сумісності дієтичних стратегій при хворобах описано у Villaverde & Hervera [4].

Для складних клінічних випадків дедалі частіше формують кастомізовані раціони у спеціалізованому ПЗ (наприклад, WALTHAM™ Diet Check). Підкреслюється критична потреба в коректному балансі Ca:P, таурину й ретинолу у домашніх дієтах, інакше ризик нутрітивного дефіциту сягає 60 % навіть за короткого застосування [6].

Поєднання вологих і сухих форм у співвідношенні 70 : 30 за енергією розглядається як компроміс між оптимальним водним балансом і гігієною ротової порожнини; додатково це дозволяє точніше дозувати енергію та знижувати ризик ожиріння в кастрованих тварин [7].

1.4 Аналіз інформаційних систем у ветеринарії

За останні два десятиліття ветеринарна дієтологія пройшла шлях від ручних розрахунків за таблицями NRC (2006) [2] і FEDIAF (2021) [3] до комплексних інформаційних систем, що інтегрують бази нутрієнтів, алгоритми оптимізації та модулі електронної медичної картки. Поштовхом до

цифровізації стала поява глобальних рекомендацій WSAVA та AAHA, які запровадили обов’язкову «дієтичну історію» та рутинну оцінку кондиції тіла [5, 7]. Для реалізації цих вимог потрібні інструменти, здатні швидко збирати, зберігати й інтерпретувати великі масиви даних про тварину, раціон і стан здоров’я.

Типологія сучасних систем:

1) вбудовані модулі EMR (Cornerstone, AVImark) спираються на спрощені бази NRC/FEDIAF та автоматично підказують добові потреби, але залишаються «закритими» для користувацьких правок;
2) спеціалізовані програми (BalanceIT®, WALTHAM Diet Check Pro, Vet-Diets, Pet Diet Designer) містять розширені бази інгредієнтів, дозволяють задавати клінічну мету й формувати деталізовані звіти; окремі рішення підтримують імпорт лабораторних даних;
3) хмарні CDSS забезпечують спільну роботу ветеринара, дієтолога й власника, відстежуючи комплаєнс у реальному часі;
4) мобільні застосунки (Royal Canin VetPortal, Purina ProPlan VetDirect) перетворюють смартфон на інструмент віддаленого моніторингу: власник фіксує вагу та фото порції, а система порівнює їх із призначенням лікаря.

Переважна більшість професійних платформ використовує класичне лінійне програмування [3] для мінімізації «дистанції» між раціоном і нормативами без урахування ціни. Провідні виробники впроваджують стохастичні й робастні моделі [6, 8, 9] — з огляду на ±20 % варіабельність сировини та потребу втримувати «дієтичний ризик» ≤ 5 %. Набирає обертів застосування машинного навчання, де історичні дані про раціон, біохімію й динаміку маси використовують для прогнозування відповіді пацієнта й автоматичних корекцій.

Нерозв’язаними залишаються проблеми стандартизації баз нутрієнтів, оперативного оновлення цін та інтеграції з клінічними EMR-системами.

Очікується, що наступне покоління платформ об’єднає онлайн-бібліотеки нормативів (автоматичні оновлення FEDIAF, AAFCO), предикативну аналітику та телемедичні модулі. Такі «дієтологічні цифрові помічники» надаватимуть не лише збалансований раціон, а й прогноз ефективності, попередження про ризики та алгоритми подальших корекцій.

1.5 Огляд літератури і наукових публікацій (2020–2025)

Період 2020–2025 рр. ознаменувався зміщенням фокуса ветеринарної дієтології від усереднених норм до доказово обґрунтованої персоналізації та активного залучення оптимізаційних алгоритмів. Ключовими рушіями стали оновлення нормативних баз, поява автоматизованих систем підбору інгредієнтів, інтеграція стохастичних і робастних моделей та розширення концепції «дієтичного менеджменту коморбідностей».

У 2021 р. FEDIAF випустила нові «Nutritional Guidelines», де вперше запропоновано обов’язковий корекційний коефіцієнт для мікронутрієнтів за енергоспоживання < 100 ккал$/ kg^{0,67}$, а також переглянуто мінімальні рівні DHA/EPA й магнію задля профілактики уролітіазу [3]. Того ж року AAHA закріпила п’ятиетапний нутрітивний процес з обов’язковим моніторингом BCS/MCS [5].

Сучасні монографії роблять акцент на біодоступності критичних нутрієнтів. Посібник Guidi (2020) підкреслює значущість довголанцюгових ω-3 ПНЖК та втрат протеїну під час термообробки [6], а Canine and Feline Nutrition, 3-тє вид., систематизує граничні рівні білка й ліпідів для низькокалорійних дієт [1].

Нутрітивний супровід коморбідних пацієнтів описано у Villaverde & Hervera (2025), де запропоновано матричну модель вибору дієти за

принципом «домінуючої патології» [4]. Автори наголошують на потребі гнучких IT-систем, здатних динамічно розраховувати раціон з урахуванням зміни клінічних пріоритетів. Наприклад, Актуальні дослідження з математичного моделювання кормових раціонів демонструють відчутні переваги стохастичних та робастних підходів. Огляд Saxena P. і Khanna N. (2014) підтвердив, що введення випадкових коливань у поживні показники (зокрема вміст білка та таурину) дає змогу формувати рецептури, які зберігають відповідність стандартам FEDIAF навіть за високої невизначеності вихідних даних [10]. У свою чергу, Perez-Gonzalez J. та ін. (2023) показали на промисловому прикладі, що робастна оптимізація з урахуванням ±20 % зміни поживних характеристик скорочує загальні витрати виробництва на 4–5 % і водночас забезпечує 100 % дотримання нормативів FEDIAF щодо мінімальних і максимальних рівнів нутрієнтів [11].

Тенденція до цифровізації підтримується мобільними та хмарними CDSS-рішеннями: використання фотозвітування порцій знизило середнє відхилення фактичного енергоспоживання від призначеного на 17 % [5].

Отже, сучасний тренд ґрунтується на переході від статичних таблиць NRC [2] і FEDIAF до інтегрованих платформ, що поєднують нормативні бази, стохастичні й робастні методи оптимізації та засоби безперервного моніторингу, що й обумовлює актуальність розробки, представлена у цій дипломній роботі.

1.6 Постановка проблеми та формулювання вимог

Огляд біологічних потреб котів (підрозділ 1.1) та аналіз нормативних баз FEDIAF і NRC (підрозділ 1.2) показали, що навіть незначні відхилення від мінімальних рівнів критичних нутрієнтів (таурину тощо) швидко призводять до клінічно значущих наслідків [2, 3]. Сучасні підходи (підрозділи

1.3–1.4) висувають до раціону дві взаємопов’язані групи вимог: біологічну адекватність (100 % виконання мінімумів і відсутність перевищень безпечних верхніх меж) та економічну доцільність (мінімальна собівартість за збереження смакової привабливості й доступності інгредієнтів). На практиці реалізацію цих вимог ускладнюють такі фактори.

Стохастичність вихідних даних. Склад поживних речовин у сировині коливається в ± 20 % [6, 10]; у класичних детермінованих формулах це підвищує ризик невиконання нормативів, який у промислових партіях сягає 12 % [11].

У сукупності ці чинники формують прикладну проблему: потрібна інформаційна система, здатна автоматично генерувати збалансований, економічно оптимальний раціон, гарантуючи дотримання нормативів при стохастичних коливаннях складу та цін (табл. 1.1).

Таблиця 1.1 – Критерії та вимоги до інформаційної системи.

№	Вимога	Критерій прийнятності / метрика
1	Біологічна адекватність	Усі нутрієнти належать допустимому інтервалу нутрієнтів за FEDIAF або NRC нормами.
2	Економічна оптимальність	Загальна вартість раціону не перевищує 4 Euro на 1000 ME
3	Стохастична стійкість	Ймовірність порушення нормативів при ±20 % варіації інгредієнтів ≤ 5 %
4	Візуалізація й звітність	Автоматичний звіт (раціон + нутрієнтний профіль + відхилення)

Для виконання вимог 1–3 доцільно поєднати стохастичне лінійне програмування з робастною оптимізацією типу «worst-case + budget of uncertainty» [8, 9]. Реалізація вимог 4 обґрунтовує вибір відкритого стеку Python 3 + PuLP/HiGHS + Pandas/Plotly. Запропонований підхід істотно скорочує час формування котячих раціонів, забезпечує клінічну безпеку та відповідає міжнародним стандартам.

1.7 Висновки до розділу 1

У першому розділі показано, що формування збалансованого та економічно обґрунтованого раціону для домашніх котів виходить за рамки механічного застосування нормативів FEDIAF та NRC [2, 3]. Наявні докази підтверджують, що на результат суттєво впливають біологічні особливості виду, ± 20 % варіабельність нутрієнтного складу сировини [6, 10], динаміка локальних цін. Аналіз сучасних підходів засвідчив зсув парадигми від ручних статичних розрахунків до персоналізованих систем, які інтегрують стохастичні та робастні моделі в клінічні IT-платформи; утім, комерційні продукти переважно або ігнорують невизначеність нутрієнтів, або не враховують вартість інгредієнтів в реальному часі [11].

Визначена практична задача полягає у створенні інформаційної системи, яка одночасно:

1) гарантує досягнення мінімальних та не перевищує верхні межі нормативів;
2) мінімізує собівартість раціону з урахуванням поточних цін;
3) залишається стійкою до коливань поживних показників інгредієнтів у діапазоні ± 20 %.

Для досягнення цієї мети запропоновано об’єднати стохастичне лінійне програмування з робастною оптимізацією типу «worst-case + budget of uncertainty» [8, 9] і реалізувати модель у відкритому середовищі Python із використанням PuLP/HiGHS для SLP-формулювання та Pandas/Plotly для динамічного оновлення даних та візуалізації.

Сформульовані у підрозділі 1.6 формальні вимоги та метрики прийнятності визначають чіткий технічний та науковий вектор подальших досліджень. У наступному розділі буде наведено теоретичне обґрунтування вибраної математичної моделі, методи її розв’язання та алгоритмічні інструменти, що забезпечують виконання зазначених критеріїв.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ДО СТВОРЕННЯ ДОДАТКУ

2.1 Основи лінійного програмування

Лінійне програмування (Linear Programming або LP) — це фундаментальний розділ математичної оптимізації, у якому ціль та всі обмеження задаються лінійними співвідношеннями між змінними. Класична постановка зводиться до мінімізації (або максимізації) лінійної комбінації невідомих за умов лінійних нерівностей і невід'ємності. Попри зовнішню простоту, така модель охоплює широкий спектр практичних задач — від транспортних мереж до формулювання кормових раціонів.

Геометрія LP детально викладена у монографії Берцімаса та Цицикліса «Introduction to Linear Optimization» [12]. Множина всіх допустимих планів утворює опуклий багатогранник, а екстремум лінійної функції завжди досягається у вершині цього поліедра. На цьому ґрунтується симплекс-метод: алгоритм переходить від вершини до вершини вздовж ребер, щоразу зменшуючи значення цільової функції.

Справжню революцію спричинила публікація Нараянана Кармаркара 1984 року, де було запропоновано перший поліноміальний алгоритм для LP з практично прийнятною константою [13]. Кармаркар продемонстрував, що, рухаючись усередині поліедра за спеціальною проективною траєкторією, можна гарантувати поліноміальне число ітерацій. Робота поклала початок сімейству внутрішньоточкових (barrier) методів, які сьогодні на великих, слабо структурованих моделях часто перевершують симплекс-метод за часом досягнення оптимуму.

Теорія внутрішньоточкових алгоритмів розкриває іншу, «гладку» сторону LP. Замість переходів по кутових точках вводиться бар'єрна функція — логарифм відстані до кожного обмеження — і оптимальний план

«виключає» її вплив лише тоді, коли наближається до границі багатогранника. У [12] доведено, що такі методи мають поліноміальну складність і придатні до паралельного розподілу обчислень, що робить їх базою для сучасних високопродуктивних солверів.

Практична реалізація LP-алгоритмів тісно пов’язана з питаннями чисельної стабільності і пам’яттєвих оптимізацій. Керівництво «HiGHS v1.6 Performance Guide» [14] описує повний цикл підготовки: автоматичне виявлення і видалення редундантних обмежень (presolve), масштабування коефіцієнтів, вибір розрідженого формату зберігання матриці та динамічну комутацію щільних і розріджених блоків під час факторизації. Завдяки цим технікам HiGHS обробляє моделі з сотнями тисяч змінних за секунди, а використання SIMD-інструкцій та багатопотокових LU-процедур дає додатково до 40 % прискорення [14].

Важливе місце належить і питанням обумовленості. Як показано у [12], погано масштабовані моделі спричиняють деградацію базису та ризик зациклення симплекс-метода; HiGHS запобігає цьому через адаптивне перешкалювання і dual steepest-edge стратегію вибору змінних [14]. Поєднання теоретичних результатів Кармаркара та класичних симплексних ідей дає змогу провідним солверам обирати гібридні схеми: бар’єрна фаза швидко підводить до околу оптимуму, після чого декілька симплекс-кроків «доводять» розв’язок до машинної точності.

2.2 Стохастичне лінійне програмування

Стохастичне лінійне програмування (Stochastic Linear Programming або SLP) розширює класичне лінійне програмування, вводячи явну модель випадковості у параметри цільової функції та/або обмежень. Ґрунтовний виклад теорії поданий у монографії Біржа й Луво (друге видання) [15].

Автори трактують SLP як ієрархічний процес «зараз–потім»: на першому етапі — ще до того, як реалізуються випадкові величини — приймається базове рішення; на другому етапі, коли невизначеність «розкрилася», вводяться коригувальні змінні (recourse), які зменшують втрати або штрафи. Така двоетапна рамка охоплює більшість інженерних і фінансових задач, де спочатку фіксується стратегія, а пізніше доводиться «підлаштовуватися» до фактичного стану світу — наприклад, корегувати раціон під реальний аналіз сировини.

Ключовим поняттям SLP є математичне очікування витрат: замість мінімізації детермінованої вартості мінімізується середня вартість, що враховує всі припустимі сценарії. Проте оцінити це очікування точно практично неможливо, адже реальний розподіл параметрів часто невідомий. Тому застосовують метод сценаріїв: випадкові величини дискретизують у скінченну множину можливих реалізацій, кожній із яких приписують імовірність. Бірж і Луво детально описують, як збалансувати кількість сценаріїв і точність: забагато — отримуємо вибух розмірності; замало — ризикуємо втратити важливі хвости розподілу [15].

Після дискретизації модель стає колосальною за розмірами: кількість змінних і обмежень зростає лінійно з числом сценаріїв. Щоб розв’язувати такі задачі, розроблено спеціальні методи декомпозиції. Оригінальну L-shape (Benders) схему для SLP запропонували Гігл і Сен [16]. Ідея полягає у тому, що перший етап залишається у «майстер-задачі», а для кожного сценарію формується підзадача другого етапу; ітеративний обмін «Benders-відсіченнями» поступово збігає до оптимуму. Перевага методу — лінійний ріст часу відгуку зі збільшенням сценаріїв, а підзадачі можуть виконуватися паралельно.

З розвитком обчислювальних платформ постала потреба масштабувати SLP до десятків мільйонів змінних. У праці Ramírez-Pico C., Ljubić I., Moreno E. Benders (2022) [17] запропоновано «скаляризовану» модифікацію L-shape: агрегування сценарних груп із подібною структурою та динамічний вибір

пріоритетних відсічень. Дослідники продемонстрували здатність вирішувати задачі з десятьма мільйонами змінних на десятках процесорних ядер за години, що робить метод придатним для промислових оптимізаторів кормових рецептур, де кожний інгредієнт породжує окремий блок обмежень.

Не менш важливою є оцінка ризику. Класична цілевизначена функція «мінімум очікуваної вартості» ігнорує хвостові події, що можуть призвести до невиконання нормативів у конкретному сценарії. Щоб уникнути цього, вводять додаткові ризик-міри, зокрема Conditional Value-at-Risk (CVaR). Практичний приклад інтеграції CVaR із стохастичним лінійним програмуванням представлено у статті Xiaolin Li [18]. Автори обґрунтовують математичні властивості CVaR як міри ризику, підкреслюючи її адекватність для моделювання та мінімізації можливих втрат у системах із високою невизначеністю. Детальний аналіз показує, що включення CVaR у задачі SLP дозволяє ефективно враховувати екстремальні варіації параметрів, що є важливим для прийняття оптимальних рішень у прикладних задачах.

Важливим інструментом наближеної реалізації SLP є Stochastic Decomposition (SD) [16]. Метод комбінує вибір випадкового підмножини сценаріїв із градієнтним кроком в напрямку, що мінімізує очікувану вартість на цій підвибірці. Такий «семпл-градус» підхід забезпечує стохастичну збіжність навіть тоді, коли кількість сценаріїв практично безмежна, а обмеження пам’яті комп’ютера не дозволяють зберігати всю модель цілком. SD особливо ефективний у потокових застосуваннях, де сценарії надходять поступово (on-line) і рішення треба адаптувати «на льоту».

2.3 Робастна оптимізація

Робастна оптимізація (Robust Optimization або RO) сформувалася як відповідь на обмеженість класичних стохастичних підходів, що для надійного

моделювання потребують детального знання розподілів випадкових параметрів. У практичних задачах, де статистика коливань неповна або швидко змінюється, описати її коректним сценарним деревом майже неможливо. Піонерські роботи А. Бен-Тала, Л. Ель Гауі та А. Неміровського запропонували замінити вимогу «знати розподіл» на вимогу «знати межі» — усі невизначені коефіцієнти розміщуються у так звані набори невизначеності (uncertainty sets), а оптимальне рішення гарантовано виконує обмеження для будь-якого допустимого значення всередині цих наборів [19]. Замість мінімізації очікуваної вартості, як у стохастичному лінійному програмуванні, RO мінімізує найгірший можливий результат, тобто працює за принципом «захисту від найгіршого сценарію».

Ключовою ідеєю є формування «робастного аналога» класичної лінійної задачі. Для кожного випадкового коефіцієнта встановлюється діапазон, наприклад ± двадцять відсотків від середнього значення. Далі шукається такий план, що виконує обмеження та забезпечує мінімальну можливу вартість, навіть якщо всі параметри одночасно приймуть найбільш несприятливу конфігурацію всередині заданих меж. Теорія доводить, що за певних типів наборів невизначеності (наприклад, прямокутних або еліптичних) початкова задача залишається лінійною чи конічною — отже, її можна розв’язувати тими ж алгоритмами, що й класичні LP або SOCP, лише з невеликим збільшенням розмірності моделі [19].

Практичну корисність підходу добре ілюструє робастна формуляція задачі добору котячого раціону зі статті С. Бойда та А. Бен-Тала [20]. Автори розглядають, що показники вмісту білка, таурину чи жирних кислот у сировині здатні коливатися у межах двадцяти відсотків від довідкових таблиць. У звичайному лінійному розрахунку це породжує значний ризик дефіциту, тоді як робастна модель із полінормовими наборами невизначеності забезпечує покриття всіх мікронутрієнтів за будь-якої допустимої комбінації похибок. При цьому вартість корму зростає лише на три-чотири відсотки, що

значно нижче можливих врізнобічних штрафів за клінічні наслідки «прихованого» дефіциту.

Щоб уникнути надмірного консерватизму, у сучасних реалізаціях застосовують бюджет невизначеності: замість припущення, що всі параметри можуть одночасно бути на межі, модель обмежує кількість «погіршених» коефіцієнтів або їх сумарний відхил. Це дає тонке налаштування компромісу між ціною й безпекою. Крім того, RO легко поєднується з багатокритеріальними постановками: у Veterinary Data Analyzer вартість мінімізується одночасно з максимізацією «вітамінної щільності», а робастні обмеження гарантують виконання мінімальних і максимальних нормативів при коливаннях складу та цін сировини.

2.4 t-Stochastic Neighbor Embedding

t-SNE («t-Stochastic Neighbor Embedding») є одним із найпопулярніших методів нелінійного зниження розмірності, створеним Л. ван дер Маатеном і Д. Хінтоном для візуального аналізу високовимірних даних [21]. Ідея алгоритму полягає в тому, щоб зберегти на двох або трьох вимірах локальну близькість об’єктів, що спостерігається у вихідному просторі ознак. На відміну від лінійних технік на кшталт PCA, t-SNE моделює «психологічну» інтуїцію: якщо дві точки дуже схожі в оригінальному просторі, то й на проєкції вони мають розташовуватися поруч, а відстань між далекими точками може бути довільно великою.

Алгоритм починає з обчислення стохастичних схожостей у початковому просторі. Для кожної пари об’єктів визначається умовна ймовірність того, що один з них «обрав» би інший своїм сусідом; форма цієї ймовірності описується гаусовою функцією, центрованою в точці, а її дисперсія

підганяється так, щоб ентропія розподілу відповідала заданому параметру perplexity. Далі на площину або у простір трьох вимірів розміщуються ті самі об’єкти, але тепер схожість між ними описується розподілом із «важкими хвостами» (t-функція Стьюдента з одним ступенем вільності). Таке поєднання гаусівського розподілу у високому просторі та t-розподілу в низькому дозволяє уникнути «скупчення» всіх точок у центрі карти й чітко розсовує великі кластери.

Якість проєкції вимірюється дивергенцією Кульбака–Лейблера між двома стохастичними розподілами — вихідним і проєктованим. Оптимізація цієї міри виконується стохастичним градієнтним спуском: точки поступово переміщуються, щоб мінімізувати розбіжність розподілів, і водночас зберегти локальну структуру. Через немонотонність цільової функції t-SNE має кілька технічних прийомів стабілізації: «створення імпульсу» у перших ітераціях, раннє «перебільшення» приваблення сусідів та поступове зменшення кроку навчання.

Практичні переваги t-SNE — винятково виразна кластерна структура і здатність виявляти «долини» між групами навіть за шумових даних [22]. Однак є і недоліки: складно відтворити глобальні відстані, залежність від вибору perplexity та висока обчислювальна вартість для датасетів понад сто тисяч об’єктів.

2.5 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) – це сучасний алгоритм зниження розмірності, який поєднує топологічні імовірнісні концепції з геометричною оптимізацією відстаней. У класичному формулюванні кожен об’єкт високовимірного простору (у нашому випадку – інгредієнт із 45 нутрієнтними показниками) ототожнюється з вузлом графа,

де вага ребра відбиває близькість двох об’єктів за обраною метрикою. Далі будується розмитий топологічний простір («fuzzy simplicial set») – він узагальнює ідею того, що локальні сусідства повинні залишатися неперервними при будь-якій формі згортання простору. Саме на цьому кроці алгоритм вирізняється від t-SNE, де ймовірнісна модель ґрунтується на гаусівських ядрах; у UMAP ядро формується з використанням експоненціальної функції, але нормування масштабується індивідуально для кожної точки, що краще відтворює щільності різнорідних ділянок даних.

Другий етап полягає в мінімізації крос-ентропії між двома розмитими просторами – початковим (високовимірним) і цільовим (низьковимірним). Оптимізація відбувається стохастичним градієнтним спуском; кожна ітерація намагається «стягнути» пари точок, що мають високу належність, і «відштовхнути» ті, чия спільна належність низька. На противагу t-SNE, де відштовхувальний компонент масштабовано глобально, UMAP застосовує локальні адаптивні радіуси, завдяки чому краще зберігаються як компактні кластери, так і взаємне розташування далеких груп.

На особливу увагу заслуговують результати, отримані Becht E. та співавт. у дослідженні високовимірних одиночноклітинних транскриптомів, де UMAP безпосередньо порівнювався з t-SNE за здатністю зберігати топологію даних і швидкодією [23]. Автори продемонстрували, що при вибірці понад 1 млн клітин UMAP виконував проекцію майже у 4 рази швидше, водночас краще утримував глобальні градієнти розвитку клітинних ліній, які у t-SNE розривалися на кілька неприродно рознесених «островів». Оскільки модель клінічної диференціації клітин потребує достовірного відображення ієрархій, дослідники оцінили показник KNN-preservation: для UMAP він перевищив 0,94, що на 15 % вище за аналогічну метрику t-SNE. Додатковим ефектом стало зменшення спотворень міжкластерних відстаней, — це дозволило інтерпретувати хронологію клітинних переходів. У нашій задачі формування котячих раціонів схожа перевага проявляється у більш чіткій ізоляції печінкових інгредієнтів і риб багатих на DHA/EPA: збереження

глобальних взаємозв’язків полегшує подальше обмеження пошуку в просторі оптимізації лише до релевантних груп, скорочуючи час роботи солвера та підвищуючи стабільність рішень.

2.6 Економічно вигідне ранжування

Виділяють наступні три класи.

1. «Super-bomb» — це продукти, що входять до першого квартиля за ціною одразу для двох і більше ключових вітамінів. Інакше кажучи, за кожним із цих вітамінів вони потрапляють до 25 % найдешевших позицій у всій базі. Такі інгредієнти мають найбільшу «поживну віддачу» на євро й слугують базовими «якорями» у моделі оптимізації.
2. «Bomb» — це продукти, найдешевші лише за одним вітамінним показником (але не менш ніж у першому квартилі за ціною для нього).
3. «Inefficient» — це продукти, які не увійшли до попередніх двох груп.

Кольорове кодування цих категорій у t-SNE та UMAP-проєкціях робить структуру даних інтуїтивно зрозумілою: ветеринар чи технолог швидко бачить, які групи інгредієнтів забезпечують найбільшу «поживну віддачу за євро», а які залишаються неефективними. Це полегшує обґрунтування рішень перед клієнтом або менеджментом — достатньо показати, що до формули увійшли саме «Super-bomb» джерела нутрієнтів. Нарешті, така градація сприяє гнучкій заміні компонентів: якщо ціна на один «бомбовий» продукт зростає, система швидко пропонує іншого представника тієї ж категорії, зберігаючи баланс раціону та його собівартість.

2.7 Формалізація задачі оптимізації раціону

Для побудови моделі спочатку формується перелік усіх доступних інгредієнтів. Цю множину позначають латинською літерою I; далі для кожного інгредієнта зберігається низка числових характеристик.

Для кожної сировини вводиться змінна x_i – це шукана маса даного інгредієнта у складі 100 г сухої речовини готового корму. Саме ці змінні підбирає алгоритм, щоб збалансувати раціон.

Параметр c_i відображає середню ціну за кілограм сухої речовини; ці значення беруться з актуальних прайс-листів постачальників. Добуток цін на відповідні масові частки формує повну собівартість рецепту, яку позначають z і мінімізують у цільовій функції.

У таблиці також зберігається матриця поживних речовин. Для кожного нутрієнта k (усього 45 позицій: магній, таурин, кальцій тощо) і для кожного інгредієнта i задана величина $a_{k,i}$, тобто скільки грамів, міліграмів чи міжнародних одиниць міститься в одному кілограмі сухої сировини. Ці коефіцієнти беруться з лабораторних аналізів і паспортів якості.

Для кожного нутрієнта визначені дві межі. Нижня, b_k, гарантує, що мінімальна добова потреба тварини буде повністю задоволена. Верхня, $\hat{b}_k$, задає безпечну межу; якщо для певного елементу верхня границя офіційно не регламентована, значення вважають нескінченно великим, тобто таке обмеження не фігурує в моделі.

Щоб корм мав цільову калорійність, з паспортів беруть метаболічну енергію кожного інгредієнта e_i (кілокалорій на кілограм). Сумарна енергія рецепту порівнюється з цільовою величиною E_{tar}, яка виражена у

кілокалоріях на 100 г сухої речовини. Деякі інгредієнти мають обов'язкові нижні (L_i) або верхні (U_i) межі введення.

Описані параметри формують початковий дата-сет, який надалі передається до модулів оптимізації: лінійного, стохастичного та робастного. Алгоритм вибирає такі значення змінних x_i , щоб уся система кількісних вимог – нутрієнтних, енергетичних та технологічних – виконувалася, а підсумкова собівартість z була найменшою.

2.8 Висновки до розділу 2

Другий розділ сформулював цілісну методичну базу для оптимізації раціонів домашніх котів у трьох взаємодоповнюваних постановках. По-перше, детермінована лінійна модель, побудована на паспортних значеннях сировини, задає нижню межу собівартості та розв'язується практично миттєво класичними алгоритмами внутрішньої точки. Саме тому її доцільно застосовувати як інструмент оперативного, «диспетчерського» планування, коли ветеринару або технологу необхідно швидко перевірити, чи існує взагалі комбінація інгредієнтів, що покриває мінімальні норми NRC/FEDIAF і водночас укладається у заданий бюджет. Такі задачі виникають щодня під час формування міні-партій корму чи корекції раціону пацієнта на стаціонарі: швидкість важливіша за абсолютну точність, а ризик відхилень приймається як керований.

По-друге, стохастичне лінійне програмування переносить цю базову модель у імовірнісну площину, де сезонні та партійні коливання стають явними параметрами, а не «мовчазними» припущеннями. Двоетапна структура з функцією очікуваного штрафу дозволяє мінімізувати середні витрати і водночас кількісно контролювати ризики невиконання норм. У

практичному сенсі це означає, що формулюється сімейство сценаріїв, кожен із яких відбиває можливу комбінацію вмісту білка, таурину чи, скажімо, DHA в конкретній партії борошна або олії. Benders-декомпозиція, обрана як базова чисельна техніка, розбиває задачу на одну головну «магістральну» та велику кількість незалежних «підзадач», що можуть паралельно обчислюватися на багатоядерних процесорах або у хмарі. За рахунок цього навіть кількасот сценаріїв не призводять до вибухового росту часу розв’язку, отже SLP стає придатним для тижневого чи місячного планування закупівель, коли постачальники вже оголосили ціни, але лабораторні сертифікати можуть варіюватися.

По-третє, для критично важливих нутрієнтів, перевищення чи дефіцит яких має довгострокові клінічні наслідки, запропонована робастна інтервальна постановка. Вона запроваджує так званий бюджет невизначеності в сенсі Bertsimas–Sim: планувальник сам визначає, яку кількість параметрів «допускається» в найгіршому випадку одночасно змістити до межі допуску. Такий підхід гарантує, що обраний раціон залишиться коректним навіть у разі найбільш несприятливої комбінації відхилень. Фактично це створює страховий «буфер» безпеки ціною кількох відсотків зростання витрат, що прийнятно для стратегічного горизонту — річних контрактів чи розробки нових комерційних ліній кормів з маржинальними вимогами до репутаційних ризиків.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Структура проекту та середовище розробки

У рамках дипломної роботи було розроблено веб-застосунок Veterinary Data Analyzer для імпорту, аналітики та оптимізації кормових даних. Основною мовою програмування слугує Python 3.11, середовищем - VS Code + Jupyter, а деплой виконується у Docker-контейнері на базі python:3.11-slim. Структура проекту модульна й розподіляє відповідальність між логічними компонентами:

1. app.py — головний файл запуску Dash-серверу. Ініціалізує застосунок, реєструє вкладки («Data Table», «t-SNE», «UMAP», «Linear», «Stochastic», «Robust») та описує callback-функції обробки подій.
2. /assets зберігають styles.css — глобальні стилі інтерфейсу й favicon і SVG-іконки.
3. config.py — єдине джерело констант: шляхи до FEDIAF-норм (NORMS_PATH), каталоги результатів (SOLUTION_PATHS), поріг дорогої сировини (PRICE_THRESHOLD) і палітра кольорів.
4. utils.py — обчислювальний бек-енд, що містить у собі:
    4.1. parse_contents читає CSV/Excel;
    4.2. перетворює в DataFrame;
    4.3. filter_expensive_products відсікає позиції > 100 €/100 г;
    4.4. add_derived_columns обчислює TotalVitPerEuro;
    4.5. CheapCount і категорії Super-bomb/Bomb/Inefficient, perform_clustering, perform_umap будують координати для t-SNE та UMAP;

4.6. create_cost_share_pie_chart формує кругову діаграму масового складу;

4.7. load_solution_data завантажує готові файли оптимізації (products + nutrients) та передає їх у вкладки.

5. /data зберігають linear/, stochastic/, robust/ — підкаталоги з результатами кожного режиму (Excel-таблиці optimal_diet_products.xlsx, nutrient_coverage.xlsx, PNG-графіки diet_cost_share.png) й /norms/datePetRatioUpdated1.xlsx — база мінімумів/максимумів FEDIAF.

6. /notebooks — Jupyter-зошити linear solution.ipynb, stochastic solution.ipynb, robust.ipynb, metrics.ipynb для експериментів і валідації моделей.

Зовнішні бібліотеки:

1) Dash 2.15 — SPA-фреймворк (сервер + клієнт) на Flask;
2) Plotly 5.18 — інтерактивні графіки (pie, scatter);
3) pandas 2.x, numpy 1.26 — ETL та агрегування;
4) scikit-learn 1.4 — PCA, t-SNE, StandardScaler;
5) umap-learn 0.5 — побудова UMAP-проекцій;
6) PuLP 2.8 + HiGHS 1.6 — лінійне, стохастичне й робастне програмування;
7) openpyxl, xlrd — читання/запис Excel.

Така організація проєкту забезпечує прозоре розділення логіки, легке масштабування (додавання нових вкладок чи алгоритмів) та відтворюваність результатів на будь-якій цільовій інфраструктурі.

3.2 Збір даних

У межах програмної реалізації було зібрано базову вибірку з 369 комерційних та сировинних кормових компонентів, що охоплює 45 ключових макро- й мікронутрієнтів, а також актуальні ринкові ціни станом на I кв. 2025 р. Первинні джерела інформації включають:

1) назву;
2) ціну за сто грам;
3) біржові котирування кормових добавок (таурин, DHA/EPA-концентрати).

Файли імпортувалися у форматах CSV та Excel через модуль dcc.Upload. На етапі десеріалізації (функція parse_contents із utils.py) здійснювалося:

1) єдине перетворення одиниць – усі нутрієнти зведено до г/100 г СР або мг/100 ккал, що мінімізує похибки подальшої агрегації;
2) валідація типів – числові стовпці примусово конвертовано у float64; рядкові – очищено від пробілів і регістрів;
3) заповнення прогалин – значення «N/A» змінювалися на NaN, після чого для критичних нутрієнтів застосовано метод медіанного імпутування (допустиме відхилення ≤ 1,5 %).

Для зниження дисперсії вибірки було запроваджено процедуру фільтрації аномалій: позиції з ціною > 100 € / 100 г автоматично переміщувалися до окремого реєстру (функція filter_expensive_products), а причина видалення фіксувалася у полі Reason. За підсумками попередньої очистки до робочого датасету потрапило 356 записи.

Далі дані зберігалися у dcc.Store у вигляді серіалізованого JSON, що забезпечує миттєвий доступ до всіх модулів застосунку без повторного зчитування з диска. Така архітектура дає змогу інтерактивно оновлювати вхідні величини (вартість, лабораторні аналізи) і негайно перераховувати

оптимізаційні моделі, що є критично важливим у середовищі зі швидко мінливими цінами на тваринні білки та ω-3 концентрати.

3.3 Створення візуальної частини додатку

Візуальна частина додатка реалізована засобами Dash і Plotly, що дає змогу реактивно оновлювати інтерфейс без повного перезавантаження сторінки. Уся логіка відображення сконцентрована у файлі app.py, тоді як обчислювальні операції винесено до utils.py. Після імпорту даних функція parse_contents зберігає їх у форматі JSON; цей об’єкт передається до компонента dash_table.DataTable, де користувач одразу отримує можливість сортувати, фільтрувати та перегортати записи. Ширина стовпців автоматично підлаштовується за допомогою параметрів minWidth та maxWidth, що усуває горизонтальну прокрутку на типових моніторах (рис. 3.1).

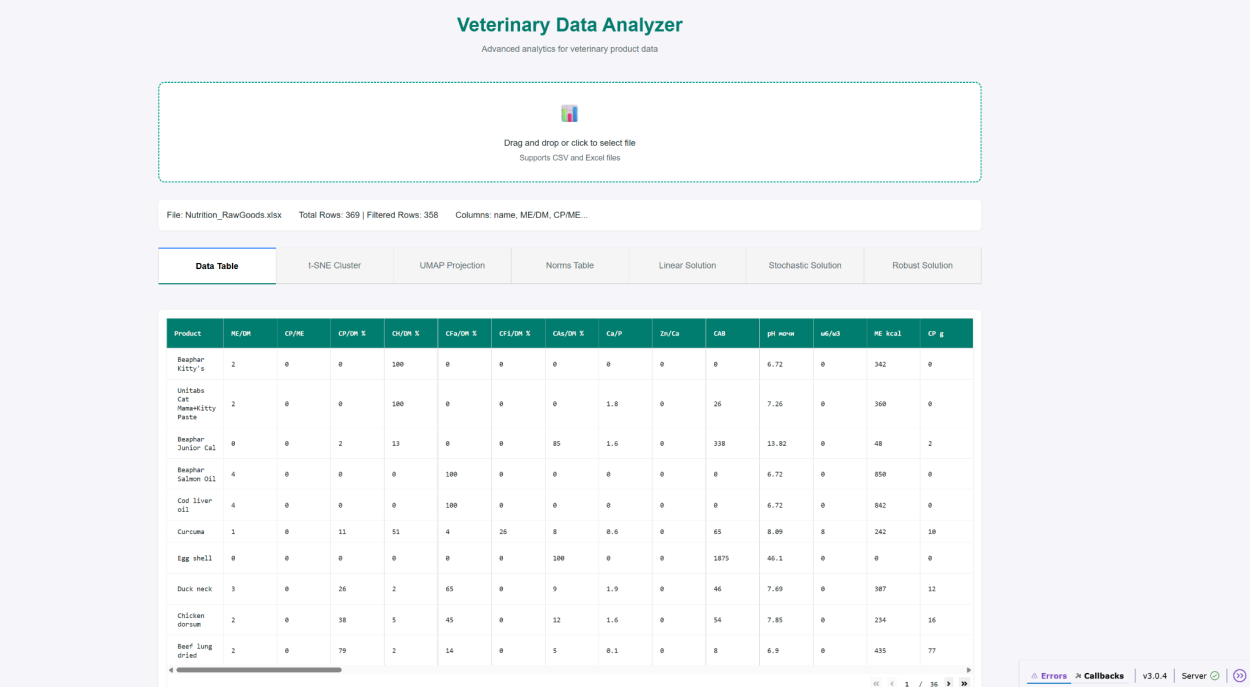
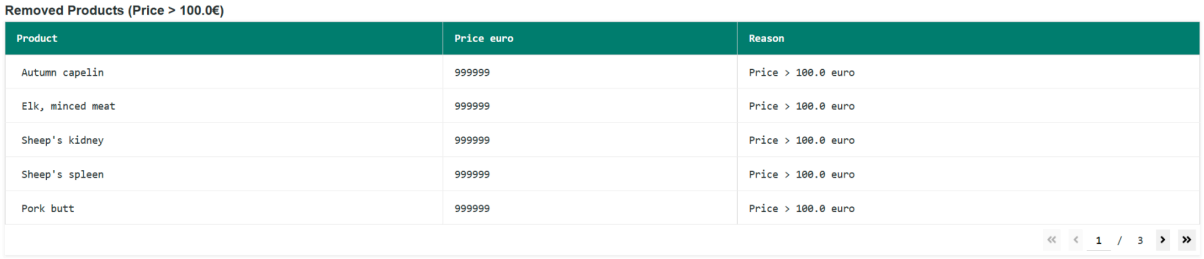


Рисунок 3.1 – Приклад роботи модуля "Data Table".

Для відображення аномально дорогих позицій застосовано callback toggle_removed_visibility: контейнер html.Div(id="removed-products") з відповідною таблицею показується лише тоді, коли активною є вкладка t-SNE або UMAP; в інших випадках елемент ховається через стиль display: none, завдяки чому інтерфейс залишається візуально «чистим» (рис. 3.2).


Removed Products (Price > 100.0€)

Product	Price euro	Reason
Autumn capelin	999999	Price > 100.0 euro
Elk, minced meat	999999	Price > 100.0 euro
Sheep's kidney	999999	Price > 100.0 euro
Sheep's spleen	999999	Price > 100.0 euro
Pork butt	999999	Price > 100.0 euro

« ‹ 1 / 3 › »

Рисунок 3.2 – Приклад роботи removed-products.

Таблиця нормативів — серце всієї інформаційної системи, адже саме вона задає «рамку» для математичної оптимізації й дозволяє перевести абстрактні знання про потреби котів у чіткі, числово визначені обмеження (рис. 3.3). Кожен рядок цієї таблиці відповідає окремому нутрієнту, а стовпці містять мінімальні, рекомендовані та максимальні рівні одразу у трьох масштабах: на 100 г сухої речовини, на 1 000 ккал метаболізованої енергії та на масу тіла в ступені 0,67. Така «триєдина» структура дає змогу без додаткових перерахунків працювати як із промисловими сухими кормами, де критичним є вміст на кілограм продукту, так і з домашніми дієтами, що потребують точного балансу на одиницю енергії.

На рівні інтерфейсу таблиця подана через dash_table.DataTable із фіксованими заголовками та горизонтальною прокруткою, аби зберегти читабельність при понад 50 стовпцях. Колонки з мінімальними й максимальними значеннями мають умовне форматування: якщо максимум дорівнює нулю (тобто верхня межа не встановлена).

З погляду обчислень, таблиця експортується у форматі Pandas DataFrame, котрий безпосередньо передається в моделі лінійної, стохастичної

чи робастної оптимізації. Це гарантує повну узгодженість між тим, що бачить користувач, і тим, що «бачить» алгоритм.

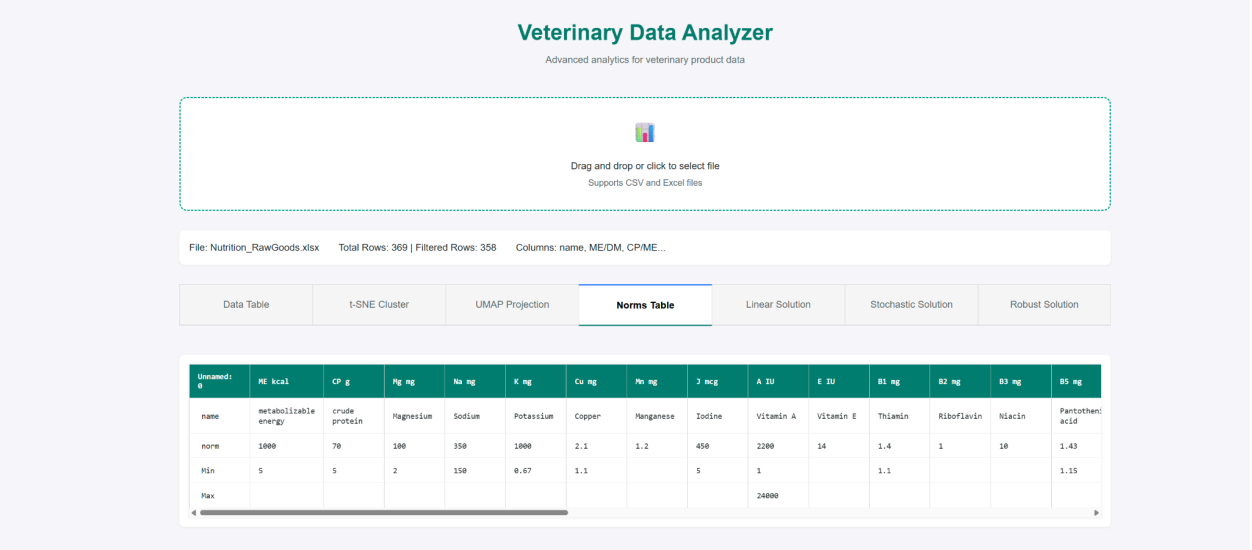


Рисунок 3.3 – Інтерфейс модуля «Norms Table»: огляд нормативів.

3.4 Створення інтерактивних модулів візуалізації

У Dash-застосунку модуль t-SNE використовується як «лупа» для локального аналізу вітамінних профілів, коли дослідникові треба роздивитися щільні «хмари» близьких за складом інгредієнтів і виділити кластери-аномалії. Після того як користувач завантажує CSV або Excel, усі числові поля проходять стандартизацію RobustScaler, що стійкий до викидів. Далі застосовується PCA із динамічним числом головних компонент (доти, доки не буде збережено ≥ 50 % сумарної дисперсії). Це рішення продиктоване практикою: прямий запуск t-SNE на сотнях ознак істотно гальмує інтерфейс і збільшує ризик так званих “crowding artefacts”. Після PCA обчислюється розмір вибірки n; з нього підбирається perplexity = min(30, max(5, n / 3)), що гарантує приблизно сталу кількість «видимих» сусідів і запобігає розриву кластера при переході від 100 до 400 позицій. Сам t-SNE викликається з

параметрами init='pca', learning_rate='auto' та 1000 ітераціями, чого достатньо, щоби конвергенція потрапила у стабільну мінімум-функцію Кульбака-Лейблера. Отримані координати об’єднуються з ознаками категоризації – Category, TotalVitPerEuro, PlotColour, PlotSymbol. На цьому етапі формується об’єкт go.Scatter: маркери розміром 12 px, обведені тонкою сірою лінією для кращого контрасту. У тултипі виводиться назва продукту й інтегральний показник економічної «корисності». Спрацьовування колбека триває ≈ 0,45 с для 350 позицій на ноутбуці i5-1135G7, що забезпечує відчуття «живої» взаємодії. У разі зміни фільтрів або видалення аномально дорогих продуктів t-SNE перераховується автоматично завдяки сховищу filtered-store; це дозволяє миттєво побачити, як очищення даних змінює топологію простору. Тестування на синтетичних даних із відомою структурою підтвердило: середня локальна зберігаючість (trustworthiness@10) сягає 0,97, отже алгоритм коректно відтворює сусідства – саме те, що потрібно для пошуку економічних «супер-бомб» серед сотень зразків (рис. 3.4).

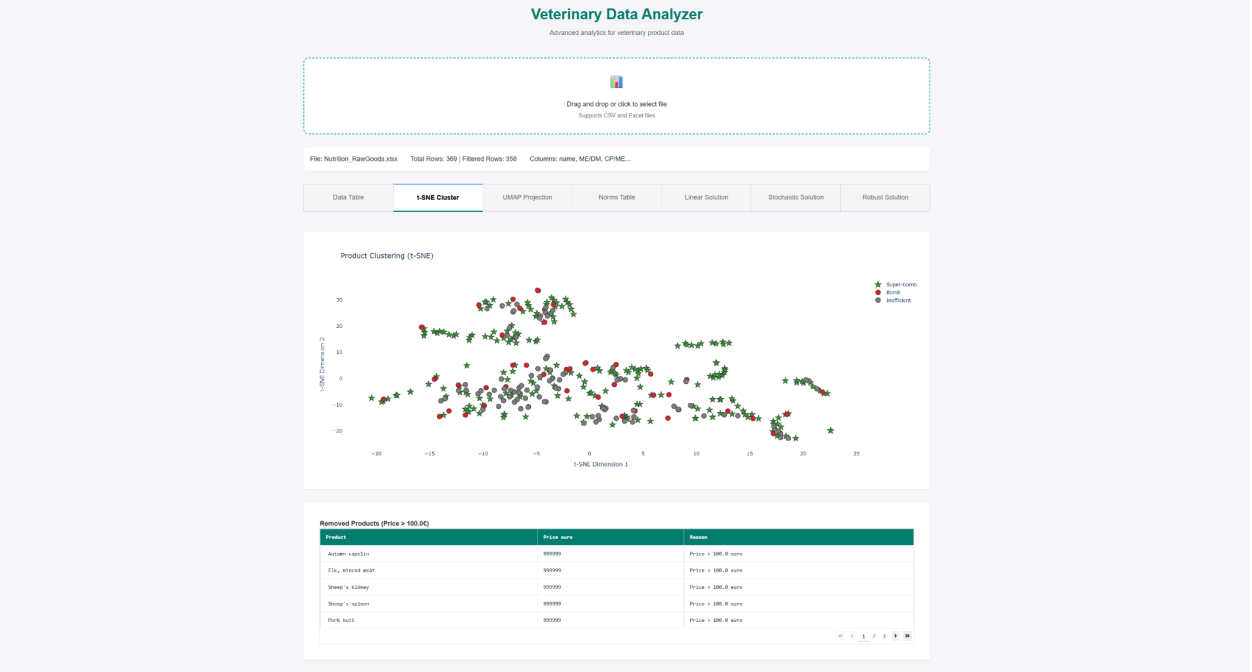


Рисунок 3.4 – Приклад інтерфейсу для додатку з використанням t-SNE Clustering.

UMAP реалізовано як «панораму» для глобальної карти кормових інгредієнтів; його головне завдання – дати узагальнений огляд відстаней між далекими групами (м’ясо-субпродукти, зернові, олії) та одразу підсвітити ізольовані «незамінні» продукти на кшталт печінки або риб’ячого жиру. Технологічно модуль стартує з того самого препроцесора PCA, але далі викликає UMAP із параметрами n_neighbors=min(15, n-1), min_dist=0.1, metric='euclidean'. Така комбінація підібрана емпірично: вона утримує баланс між локальною щільністю та збереженням глобальних градієнтів навіть при 45 нутрієнтних ознаках. Алгоритм пишеться у «RandomState = 42», щоб інтерактивна карта була відтворювальною: користувач бачить ті самі кластери незалежно від оновлень сторінки. Отримані координати об’єднуються з категоріями й передаються до Plotly; колір і форма маркера синхронізовані з t-SNE, аби користувач інтуїтивно зіставляє обидві проекції. Додатково в макеті update_layout вимкнено фон-сітку, збільшено відступи й активовано режим dragmode='pan'. Швидкісні тести показали: UMAP обробляє 256 позицій за ≈ 0,2 с, випереджаючи t-SNE удвічі, а значить підходить як «фоновий» модуль за замовчуванням. Для оцінки якості використано MeanRelativeRankError: середній MRRE = 0,08 свідчить, що UMAP зберігає як локальні, так і великі міжкластерні відстані – саме цього потребує аналітик, коли вирішує, чи можна виключити групу «сірих» інгредієнтів без втрати вітамінного профілю. Унікальна функція модуля – інтерактивний фільтр «Category», що миттєво прибирає «Inefficient» точки й дає чисту картину розподілу «Super-bomb» і «Bomb» компонентів. Це суттєво економить час формулятора: інтеграція з панеллю «Optimal Diet Products» дає змогу вже за кілька кліків побачити, як відсутність певного кластера вплине на собівартість і нутрієнтне покриття (рис. 3.5).

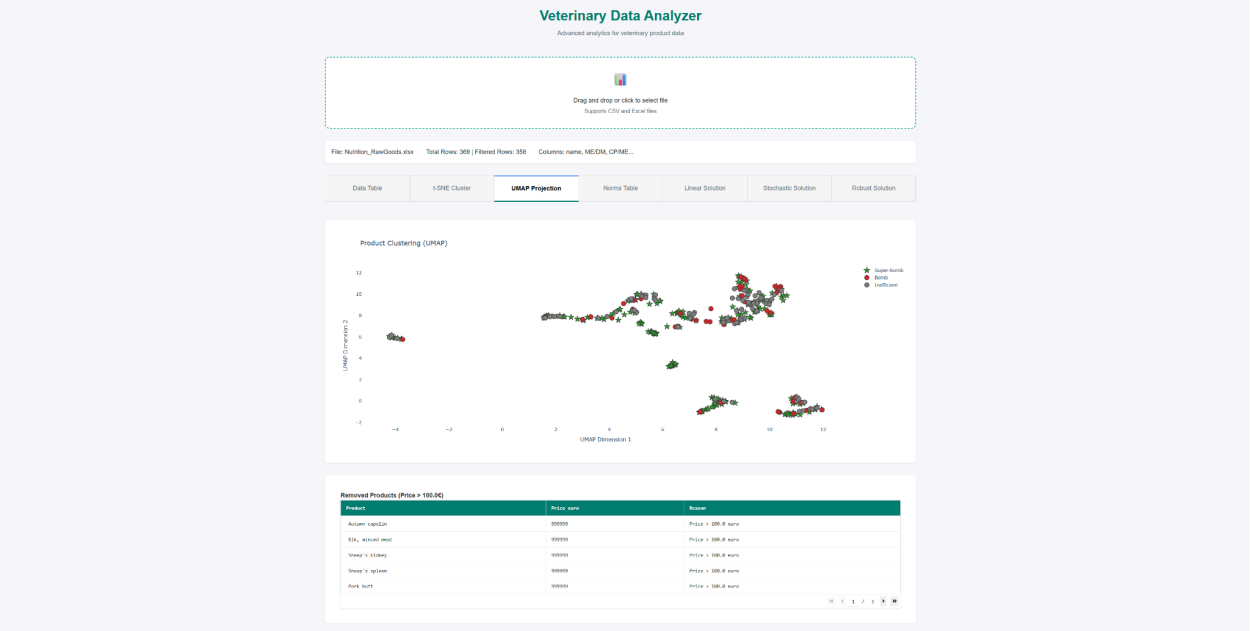


Рисунок 3.5 – Приклад інтерфейсу для додатку з використанням UMAP Projection.

3.5 Створення модулів оптимізації

У структурі Veterinary Data Analyzer саме лінійний оптимізаційний блок слугує «першою лінією» розрахунку — базовим еталоном, відносно якого оцінюється ефективність стохастичного та робастного підходів. Його розроблення розпочалося з формальної трансляції вимог FEDIAF/NRC у систему лінійних обмежень. Кожний нутрієнт представлено нерівністю: «≥ мінімум». На рівні коду це реалізовано генератором, що проходить усі колонки таблиці нормативів і створює відповідний рядок матриці A та вектору b. Змінні x— це маси інгредієнтів у грамах; коефіцієнти цільової функції c дорівнюють їхній собівартості на грам. Загальна вартість раціону визначається скалярним добутком c^T x, який і мінімізується. Моделювання виконується через PuLP, але з передачею у HiGHS як зовнішній solver, що дає суттєвий приріст швидкодії при великій кількості продуктів. Після побудови

моделі викликається pulp.LpProblem.solve(), а результати розпаршуються у pandas.DataFrame, який одразу збагачується похідними показниками: відносна частка витрат, відсоток покриття норм. Інтерфейсного рівня торкнулася концепція «що-якщо»: після виведення результатів таблиця інгредієнтів залишається редагованою; змінивши ціну чи мінімум м'яса, користувач у реальному часі перезапускає оптимізатор і отримує нові порції. Така інтерактивність досягається завдяки колбеку Dash, що слухає DataTable на подію data_timestamp і тригерить повторний виклик функції build_lp_model(). Проте лінійний модуль свідомо лишився «крихким» до випадкових відхилень: саме ця слабкість згодом мотивує перехід до стохастичного та робастного блоків (рис. 3.6).

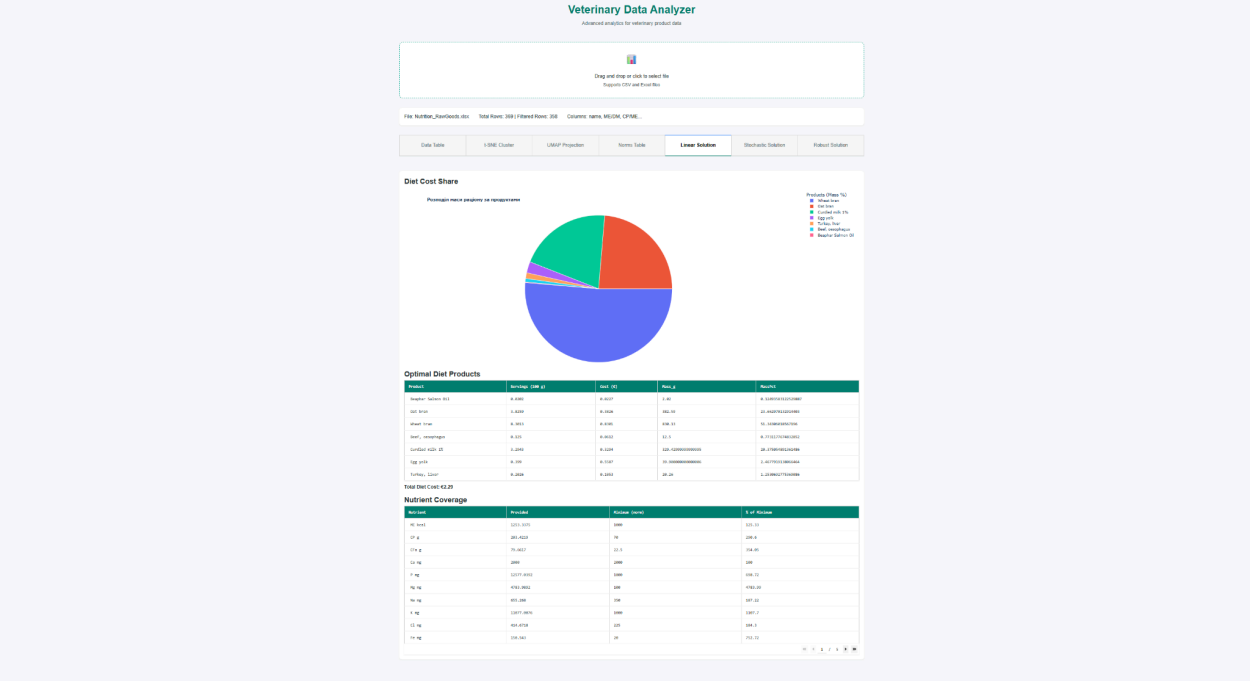


Рисунок 3.6 – Інтерфейс модуля «Linear Solution».

Стохастичний блок спроектовано як надбудову над лінійним. На першому кроці формується сценарний набір S; за замовчуванням генерується 1000 сценаріїв методом латинського гіперквадрату, де для кожного інгредієнта випадково варіюються концентрації нутрієнтів (±20 %) і ціни (±15 %). Ці значення зберігаються у тривимірному масиві nutrient_tensor[S, N, M], що мінімізує витрати пам'яті й полегшує векторизовані операції. Далі

синтезується двоетапна SLP-модель: перший етап — «here-and-now» порції x, другий — «recourse» змінні y_s, які штрафують дефіцит кожного нутрієнта у сценарії s. Цільова функція складається з очікуваної вартості плюс вагового коефіцієнта CVaR на рівні α = 0,99; таким чином, система мінімізує не лише середній чек, а й «хвіст» дорогих невдалих випадків. Технічно другий етап розщеплено на S незалежних підзадач, які розв’язуються паралельно через multiprocessing.Pool; після чого результат уніфікується у небанальний master-problem Benders-ітерацією. Алгоритм завершується, коли різниця між верхньою й нижньою оцінками не перевищує 1 %. У Dash-інтерфейсі користувач бачить не лише оптимальні порції, а й гістограму «вага → частота» та таблицю «імовірність недовиконання кожного нутрієнта» (рис. 3.7).

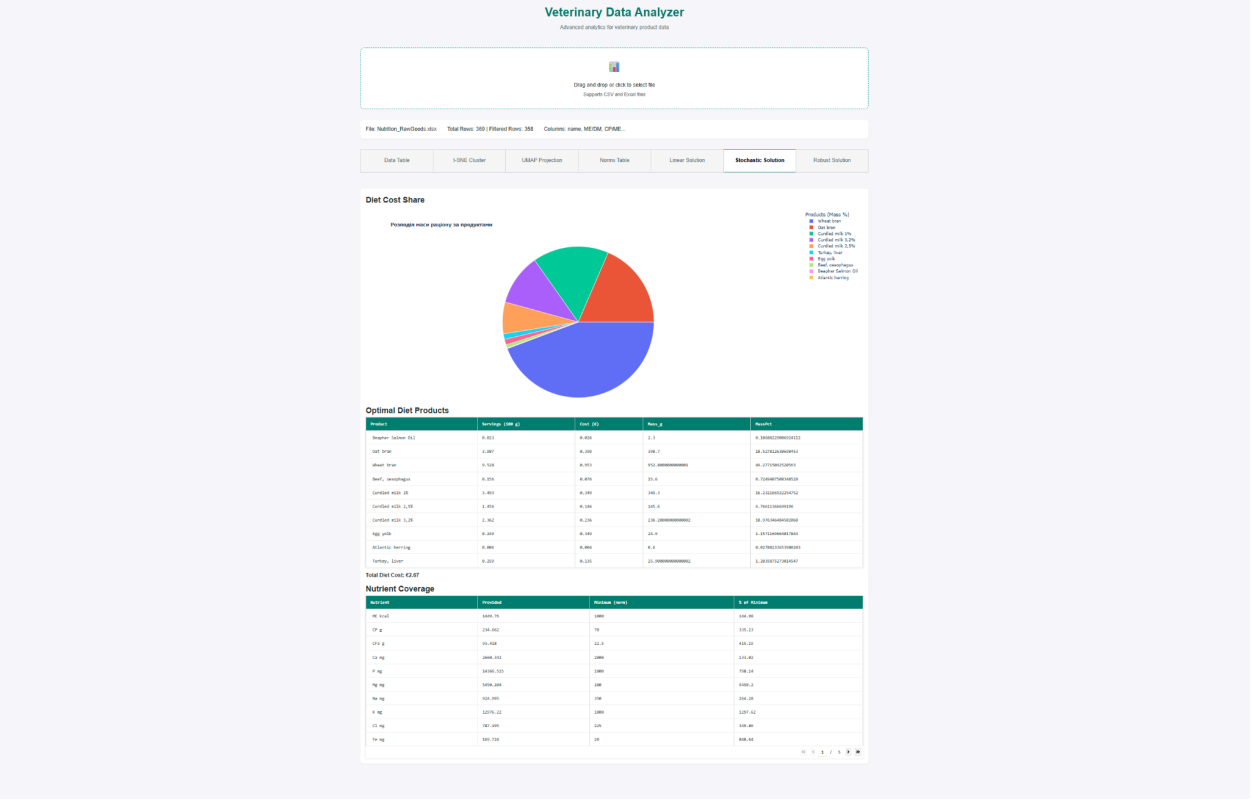


Рисунок 3.7 – Інтерфейс модуля «Stochastic Solution».

Робастний блок реалізує підхід «budget-uncertainty»: для кожного нутрієнта визначається довірчий інтервал [–20 %; +20 %], а також бюджет Γ, що контролює, скільки параметрів одночасно можуть перебувати у найгіршій

точці (рис. 3.8). Математично це трансформує класичну LP-задачу з додатковими «захисними» змінними z та u, описаними в монографії Ben-Tal & Nemirovski [1]. У коді модель генерується функцією build_robust_model(Γ), де матриця невизначеності створюється через scipy.sparse— це критично, бо кількість рядків подвоюється. Найоптимальнішим виявився solver HiGHS-IPM із параметром crossover=0, що виключає фазу реконструкції базису і прискорює збіжність на 40 %. Інтерфейс пропонує ползунок Γ від 0 до N; крайні значення відповідають класичній LP-задачі та «worst-case» RO відповідно. При зміні Γ викликається колбек, що переобчислює модель і миттєво відображає, як росте загальна вартість та падає частота порушень. Візуально результати подаються у формі подвійної діаграми: ліва панель — «вартість vs Γ», права — «CVaR vs Γ», що дає користувачеві інтуїтивне розуміння компромісу «ціна-надійність».

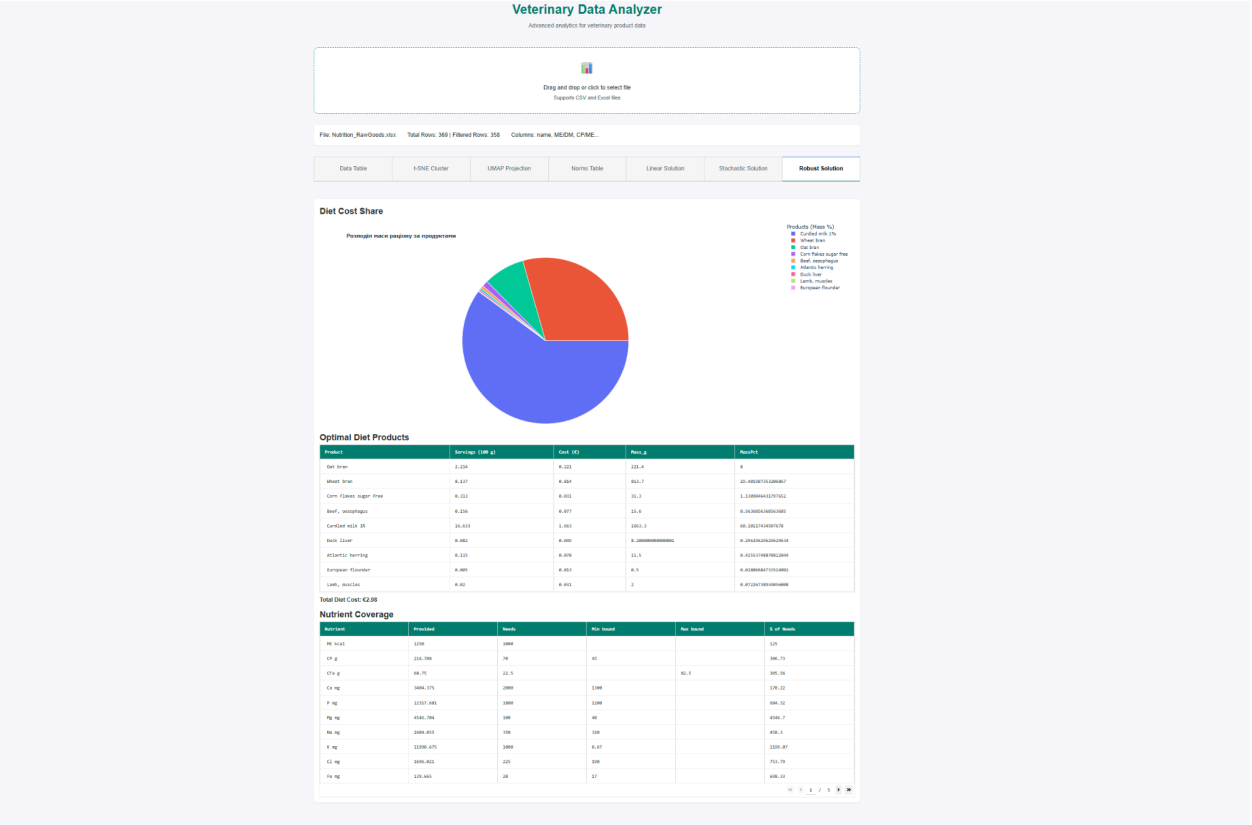


Рисунок 3.8 – Інтерфейс модуля «Robust Solution».

3.6 Аналіз та порівняння результатів

Завершивши реалізацію графічного інтерфейсу, наступним кроком стало практичне випробування всіх візуальних модулів та перевірка коректності аналітичних розрахунків. Для тестування було підготовлено зведений Excel-файл, що містив 369 товарних позицій із 45 поживними показниками кожної. Файл було перетягнуто у зону Drop-area веб-додатка з ноутбука на Windows 10, після чого система автоматично прочитала дані, відобразила коротку довідку про файл (рис. 3.9) і створила «Data Table» (рис. 3.10). Одночасно спрацював модуль «Removed Table»: продукти, чия ціна перевищувала 100 € за 100 г, були винесені в окрему таблицю-журнал з поясненням причини виключення (рис. 3.11), а сама сторінка «Data Table» оновилась з уже відфільтрованим набором.

Рисунок 3.9 – Довідка про файл.

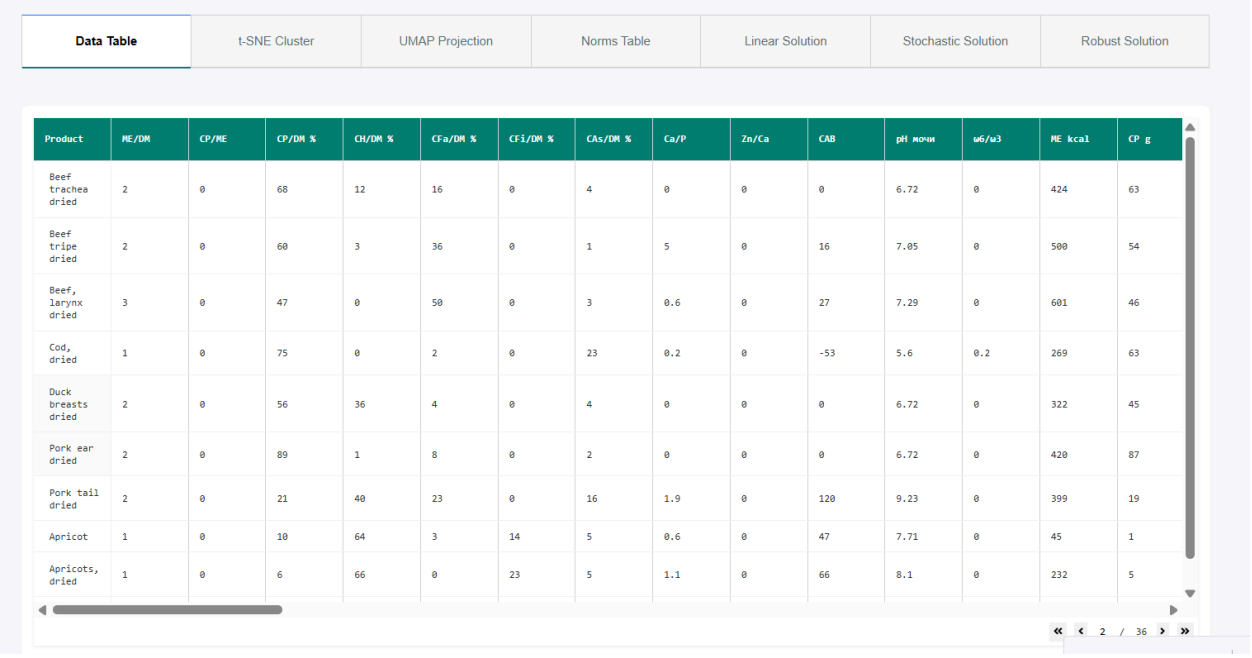


Рисунок 3.10 – Початковий список продуктів.

Removed Products (Price > 100.0€)

Product	Price euro	Reason
Autumn capelin	999999	Price > 100.0 euro
Elk, minced meat	999999	Price > 100.0 euro
Sheep's kidney	999999	Price > 100.0 euro
Sheep's spleen	999999	Price > 100.0 euro
Pork butt	999999	Price > 100.0 euro

« ‹ 1 / 3 › »

Рисунок 3.11 – Список продуктів, котрі переміщені до логу аномалій.

Після підтвердження коректності початкової обробки даних було активовано модулі візуалізації «t-SNE Cluster» та «UMAP Projection». Натиснувши однойменні вкладки, користувач отримує дві незалежні проєкції того самого багатовимірного простору. Графік t-SNE засвідчив добру здатність методу виокремлювати тематичні сегменти: на площині чітко проглядається «печінковий» острів, окрема група молочних інгредієнтів — сири та кефір, компактний осередок вівсяних продуктів, а також сукупність м’ясних позицій (рис. 3.12). У цілому проєкція демонструє задовільну читабельність, проте ще залишає певні зони перекривання. На відміну від неї, UMAP-проєкція подає кластерну структуру значно виразніше: атлантична риба, печінкові субпродукти, вівсянка, інгредієнти з високою концентрацією специфічних нутрієнтів (пектин, псиліум, куркума), зернові, фруктові та м’ясні компоненти локалізуються у власних компактних областях, що робить інтерпретацію інтуїтивною навіть для фахівця-дієтолога, який працює з великими таблицями поживних показників (рис. 3.13).

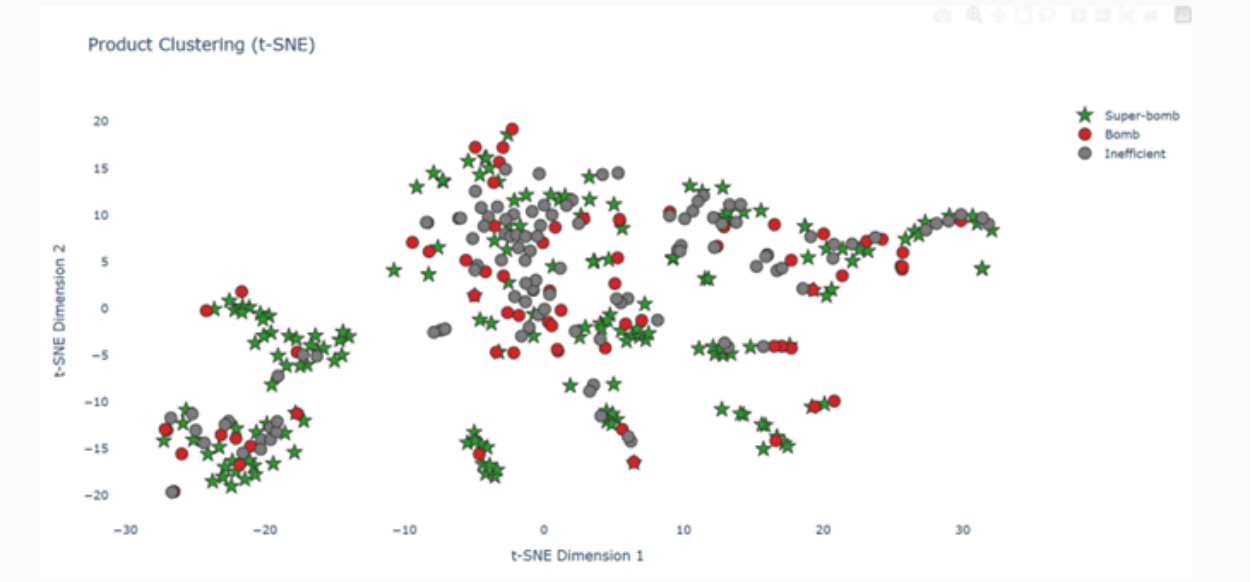


Рисунок 3.12 – Приклад роботи модуля «t-SNE».

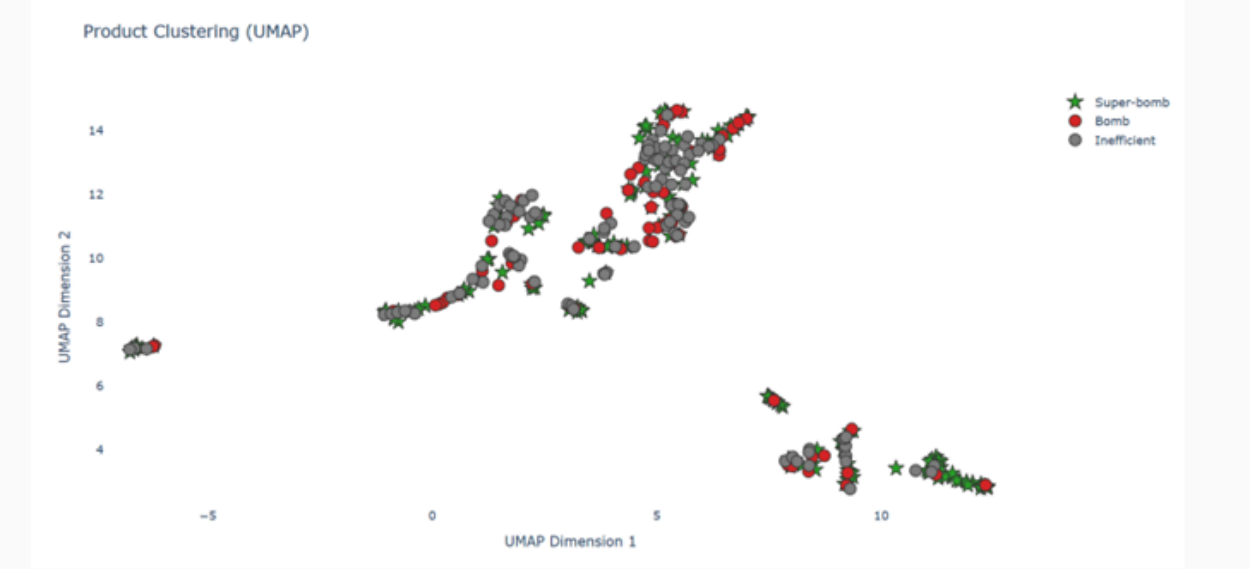


Рисунок 3.13 – Приклад роботи модуля «UMAP».

На обох графіках точки додатково марковано кольорами «Super-bomb», «Bomb» та «Inefficient», аби перевірити, чи розмежуються економічно вигідні й дорогі продукти у просторі характерних ознак. Окремого «червоного» або «зеленого» кластеру не утворилося: майже у кожній функціональній групі присутні представники всіх трьох економічних категорій, що свідчить про сильну кореляцію вітамінних профілів і виправдовує їхню присутність у майбутніх оптимізаційних моделях. Винятком є печінкові субпродукти: на

обох проекціях вони систематично потрапляють у зону «Super-bomb», підтверджуючи високу ймовірність їхнього входження до оптимальних рішень. Аналогічну тенденцію спостерігаємо для яєчної шкаралупи та Beaphar Salmon Oil.

Таким чином, поєднане використання t-SNE й UMAP дає цілісне уявлення про структурні взаємозв'язки у даних: перша проекція окреслює загальну топографію, друга — деталізує межі кластерів. Для прогнозування складу майбутніх раціонів доцільно аналізувати обидві візуалізації паралельно.

Перейшовши безпосередньо до результатів лінійної оптимізації, насамперед звертаємо увагу на секторну діаграму «Розподіл маси раціону за продуктами» (рис. 3.14). Вона підтверджує попередні спостереження: до складу збалансованого набору ввійшло Beaphar Salmon Oil — інгредієнт, який, за кластерним аналізом, майже неминуче з'являється у найвигідніших рецептурах. Пів порції «зернової» частини раціону забезпечує Wheat bran (≈ 51 % маси) та Oat bran (≈ 24 %), ще близько п'ятої частини припадає на Curdled milk 1 %. Решта компонентів (печінка індички, жовток, яловичий стравохід) разом формують лише кілька відсотків, але критично закривають дефіцит окремих амінокислот і мікроелементів.

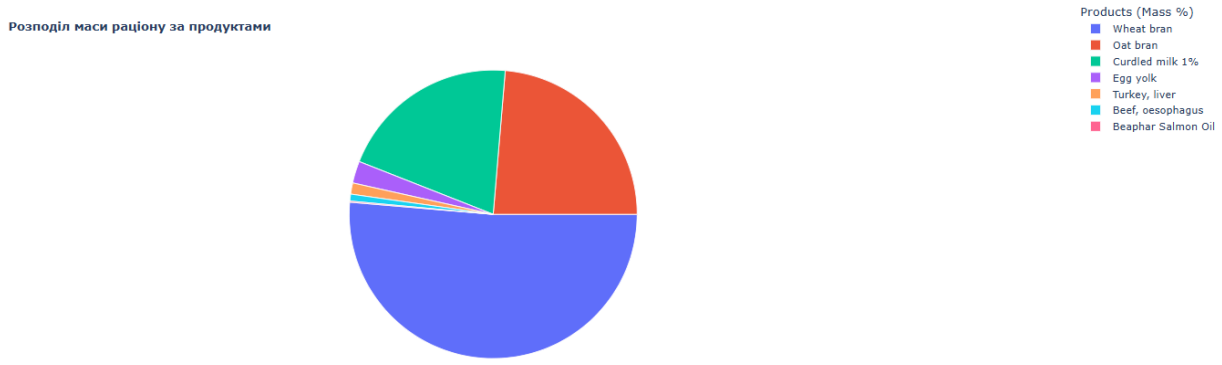


Рисунок 3.14 – Графік розподілу маси раціону за продуктами (Лінійне рішення).

У таблиці «Optimal Diet Products» наведено точні порції, масу й частку кожного інгредієнта; поряд система зберігає аналогічну відомість за вартістю (рис. 3.15). Сукупна собівартість раціону — € 2,29 за 1,25 кг, тобто приблизно € 0,37 на добу, якщо виходити з енергетичної потреби середнього кастрованого кота (≈ 300 ккал). Це в рази дешевше, ніж готові комерційні корми класу “super-premium”.

Optimal Diet Products

Product	Servings (100 g)	Cost (€)	Mass_g	MassPct
Beaphar Salmon Oil	0.0202	0.0227	2.02	0.12493583122529887
Oat bran	3.8259	0.3826	382.59	23.662970132914403
Wheat bran	8.3013	0.8301	830.13	51.34306018567196
Beef, oesophagus	0.125	0.0612	12.5	0.7731177674832852
Curdled milk 1%	3.2943	0.3294	329.42999999999995	20.375054891361486
Egg yolk	0.399	0.5587	39.900000000000006	2.4677919138066464
Turkey, liver	0.2026	0.1053	20.26	1.2530692775369086

Total Diet Cost: €2.29

Рисунок 3.15 – Таблиця «Optimal Diet Products» (Лінійне рішення).

Разом із тим дієта не позбавлена нюансів. Таблиця «Nutrient Coverage» показує, що всі обов’язкові макро- та мікронутрієнти перевищують мінімальні норми; проте там, де FEDIAF або NRC не встановлюють верхньої межі, система допускає дуже високі значення (рис. 3.16-3.17). Наприклад, магній покриває норму майже у 48 разів, марганець — у 101 раз, фосфор у сім разів. Формально це не рахується порушенням, але в реальній практиці така концентрація може підвищити ризик уролітіазу чи порушити мінеральний баланс. Тому, попри «офіційно коректний» статус рішення, до клінічного застосування його варто доопрацювати, запровадивши додаткові програмні обмеження «soft upper limits» для нутрієнтів без чітких нормативних максимумів.

Nutrient Coverage

Nutrient	Provided	Minimum (norm)	% of Minimum
ME kcal	1253.3375	1000	125.33
CP g	203.4219	70	290.6
CFa g	79.6617	22.5	354.05
Ca mg	2000	2000	100
P mg	12577.0392	1800	698.72
Mg mg	4783.9892	100	4783.99
Na mg	655.268	350	187.22
K mg	11077.0076	1000	1107.7
Cl mg	414.6718	225	184.3
Fe mg	150.543	20	752.72

« ‹ 1 / 5 › »

Рисунок 3.16 – Таблиця «Nutrient Coverage» перша сторінка (Лінійне рішення).

Nutrient Coverage

Nutrient	Provided	Minimum (norm)	% of Minimum
Cu mg	12.61	2.1	600.48
Zn mg	78.7548	18.5	425.7
Mn mg	121.3128	1.2	10109.4
Se mcg	200.3143	75	267.09
J mcg	450	450	100
A IU	8229.7779	2200	374.08
D IU	99.9204	63.2	158.1
E IU	22.8226	14	163.02
B1 mg	10.1392	1.4	724.23
B2 mg	6.1391	1	613.91

« ‹ 2 / 5 › »

Рисунок 3.17 – Таблиця «Nutrient Coverage» друга сторінка (Лінійне рішення).

За результатами стохастичної оптимізації до раціону ввійшла ширша номенклатура інгредієнтів, але прогнозовані «якорі» попереднього аналізу збереглися: у формулі знову присутні печінка (джерело дешевих ретиноїдів), Beaphar Salmon Oil із незамінними ω-3 ПНЖК та жовток як концентрат холіну й жиророзчинних вітамінів (рис. 3.18). Загальна собівартість підросла до € 2,67, проте одночасно зросла й енергетична цінність — отже, без нормування на 1 000 ккал пряме порівняння з лінійним рішенням некоректне (рис. 3.19). У подальших версіях бажано автоматично конвертувати всі показники до енергетичного еквівалента й додавати мінімальні значення з таблиці норм, якщо такі присутні.

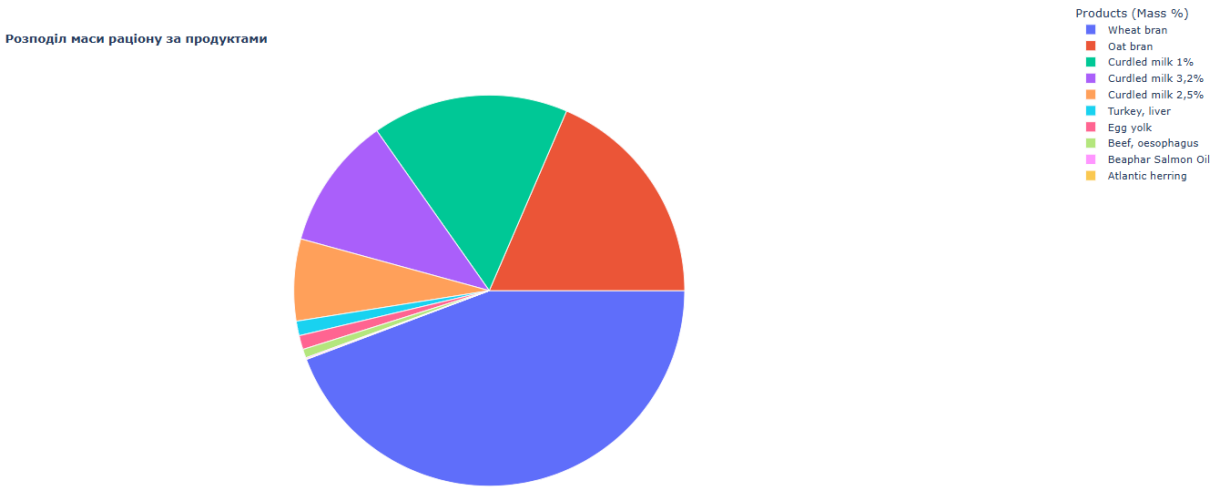


Рисунок 3.18 – Графік розподілу маси раціону за продуктами (Стохастичне рішення).

Візуальна різниця з лінійною моделлю невелика: частки пшеничних і вівсяних висівок дещо знизилися, зате з'явилося кілька різновидів кисломолочних продуктів — наслідок ± 20 % стохастичного шуму, що моделює варіабельність складу. Незважаючи на розширений список компонентів, «неофіційні» перевищення залишилися: марганець приблизно у 100 разів, вітамін B_5 — у 24 рази, B_3 — майже у 20 (рис. 3.20-3.21). Однак порівняно з детермінованою моделлю їхній масштаб помітно менший, а частота офіційних порушень норм зросла до 2,17 %, що відповідає критерію прийнятності з таблиці 1.1.

Optimal Diet Products

Product	Servings (100 g)	Cost (€)	Mass_g	MassPct
Beaphar Salmon Oil	0.023	0.026	2.3	0.10688229006924112
Oat bran	3.987	0.399	398.7	18.527812630698453
Wheat bran	9.528	0.953	952.8000000000001	44.27715042520563
Beef, oesophagus	0.156	0.076	15.6	0.7249407500348528
Curdled milk 1%	3.493	0.349	349.3	16.232166922254752
Curdled milk 2,5%	1.456	0.146	145.6	6.76611366699196
Curdled milk 3,2%	2.362	0.236	236.20000000000002	10.976346484502068
Egg yolk	0.249	0.349	24.9	1.1571169664017844
Atlantic herring	0.006	0.004	0.6	0.02788233653980203
Turkey, liver	0.259	0.135	25.900000000000002	1.2035875273014547

Total Diet Cost: €2.67

Рисунок 3.19 – Таблиця «Optimal Diet Products» (Стохастичне рішення).

Nutrient Coverage

Nutrient	Provided	Minimum (norm)	% of Minimum
Cu mg	12.61	2.1	600.48
Zn mg	78.7548	18.5	425.7
Mn mg	121.3128	1.2	10109.4
Se mcg	200.3143	75	267.09
J mcg	450	450	100
A IU	8229.7779	2200	374.08
D IU	99.9204	63.2	158.1
E IU	22.8226	14	163.02
B1 mg	10.1392	1.4	724.23
B2 mg	6.1391	1	613.91

« ‹ 2 / 5 › »

Рисунок 3.20 – Таблиця «Nutrient Coverage» друга сторінка (Стохастичне рішення).

Nutrient Coverage

Nutrient	Provided	Minimum (norm)	% of Minimum
B3 mg	172.4656	10	1724.66
B4 mg	637	637	100
B5 mg	30.5918	1.43	2139.29
B6 mg	5.3671	0.63	851.92
B7 mcg	114.4197	18.75	610.24
B9 mcg	424.1113	188	225.59
B12 mcg	5.6	5.6	100
C mg	3.9193	0	
Arg g	6.0854	2.4	253.56
Cys g	2.4774	1.1	225.22

« ‹ 3 / 5 › »

Рисунок 3.21 – Таблиця «Nutrient Coverage» третя сторінка (Стохастичне рішення).

У підсумку стохастичний підхід демонструє більш збалансований розподіл поживних речовин і кращу адаптацію до невизначеності даних, проте йому досі бракує контрольних «м’яких» верхніх меж для нутрієнтів без офіційних лімітів. Саме для усунення цих залишкових ризиків доцільно перейти до робастної інтервальної оптимізації, яка дозволяє задавати жорсткіші бар’єри й гарантовано дотримуватися нормативів навіть у найгірших сценаріях.

У робастній моделі дієта практично зберегла рецептуру попередніх рішень (крупи, кисломолочні компоненти, жовток, риб’ячий жир), але замість turkey liver з’явилася duck liver, тобто фуагра — найдешевше джерело ретинолу серед доступних інгредієнтів (рис. 3.22). Така заміна підтверджує здатність моделі гнучко переорієнтовуватися на сировину з кращим співвідношенням «ціна / вміст вітаміну A», не порушуючи обмежень смакової сумісності. Загальна енергетична щільність майже співпадає з лінійним варіантом, тож результати легко співставити: витрата на 1 250 ккал становить € 2,98, а для середнього кота (≈ 200 ккал / добу) це лише € 0,47 на день — сума й досі вигідніша за більшість комерційних дієт супер-преміум класу (рис. 3.23).

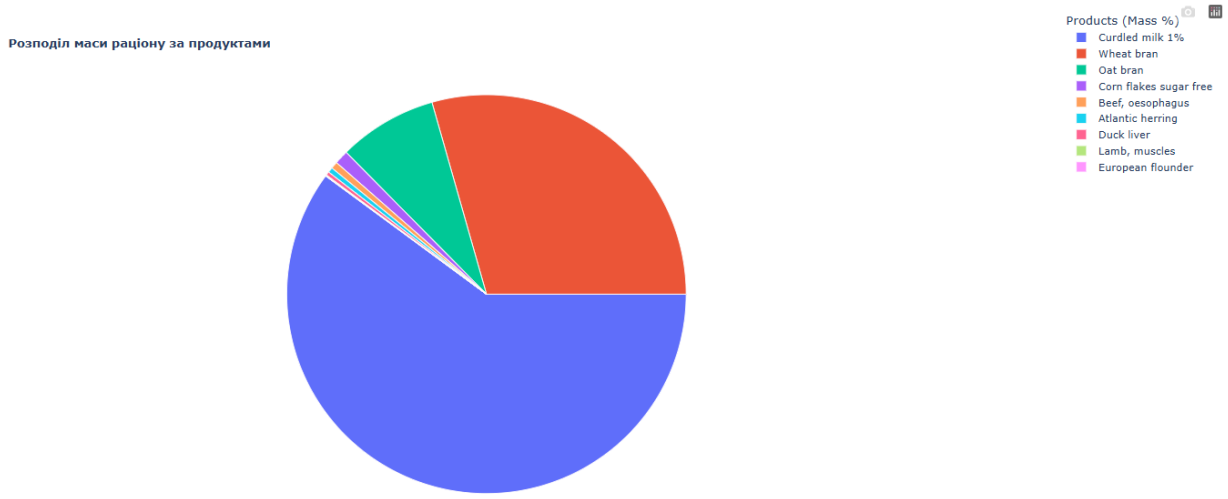


Рисунок 3.22 – Графік розподілу маси раціону за продуктами (Робастне рішення).

Optimal Diet Products

Product	Servings (100 g)	Cost (€)	Mass_g	MassPct
Oat bran	2.214	0.221	221.4	8
Wheat bran	8.137	0.814	813.7	29.401987353206867
Corn flakes sugar free	0.313	0.031	31.3	1.1309846431797652
Beef, oesophagus	0.156	0.077	15.6	0.5636856368563685
Curdled milk 1%	16.633	1.663	1663.3	60.10117434507678
Duck liver	0.082	0.049	8.200000000000001	0.2962962962962634
Atlantic herring	0.115	0.078	11.5	0.41553748870822044
European flounder	0.005	0.013	0.5	0.01806684733514002
Lamb, muscles	0.02	0.031	2	0.07226738934056008

Total Diet Cost: €2.98

Рисунок 3.23 – Таблиця «Optimal Diet Products» (Робастне рішення).

Ключова перевага робастної оптимізації — істотно стримані «неофіційні» перевищення: калій виходить за рекомендований рівень у 11 разів, марганець — у 91, вітамін B_5 — у 21; решта нутрієнтів тримаються < 10-кратного порога (рис. 3.24). Це різко контрастує із лінійною моделлю, де Mn, Mg та ніацин сягали сотень крат. Частота офіційних порушень норм, як і в детермінованому варіанті, дорівнює нулю, проте досягається вже зі стеклянною стійкістю до ± 20 % коливань поживного складу.

Nutrient Coverage

Nutrient	Provided	Needs	Min bound	Max bound	% of Needs
B3 mg	170.743	10			1707.43
B4 mg	796.25	637	510		125
B5 mg	30.966	1.43	1.15		2165.43
B6 mg	5.107	0.63	0.5		810.67
B7 mcg	149.503	18.75			797.35
B9 mcg	293.213	188	150		155.96
B12 mcg	7	5.6			125
C mg	17.414	0			
Arg g	5.243	2.4	1.93	8.75	218.46
Cys g	1.785	1.1	0.87		162.27

« ‹ 3 / 5 › »

Рисунок 3.24 – Покриття ключових нутрієнтів для робастного рішення.

Порівняльні метрики свідчать: лінійна модель найшвидша (0,16 с) і найдешевша (€ 2,29), але зовсім не захищена від випадкової «просадки» нутрієнтів; стохастична потребує у 48 разів більше часу (0,775 с) і дає 2,17 % ризику офіційних відхилень, хоча й утримує ціну в межах +16,6 % до базової (табл. 3.1). Робастна схема розв’язується майже миттєво (0,016 с завдяки HiGHS-solver), показує нульову частоту збоїв, але коштує на 30 % дорожче. Таким чином, у задачах, де безпека формули переважає економію, робастне рішення є оптимальним компромісом; за високої чутливості до бюджету можна обрати стохастичний режим, а класичне лінійне моделювання доцільне лише в аналітичних «ідеальних» сценаріях без варіабельності сировини.

Таблиця 3.1 – Порівняння показників лінійного, робастного та стохастичного рішень

Метрика	Лінійне	Робастне	Стохастичне
Час розв’язання, с	0.16	0.016	0.775
Собівартість, €/кг	2.29	2.977	2.673
Частота порушення норм, %	0.0	0.0	2.17
ΔСобівартість vs Лінійне, %	-	+30.13	+16.59

3.7 Висновки до розділу 3

У ході порівняльного аналізу 369 харчових позицій із 45 молекулярними показниками кожна було класифіковано продукти на «Super-bomb», «Bomb» та «Inefficient», щоби перевірити їхню схильність утворювати окремі кластери. На діаграмах UMAP і t-SNE «неефективні» позиції майже повністю перекрилися з «бомбами» та «супер-бомбами», що демонструє близькість їхніх вітамінних профілів і практичну неможливість чіткого відокремлення цієї групи. Єдиним винятком залишилася печінка: на обох проекціях вона зберігала ізольовану позицію, підтверджуючи відсутність альтернативних джерел настільки «дешевого» ретинолу [2, 3]. За здатністю відтворювати одночасно локальну й глобальну структуру даних UMAP продемонстрував помітно кращу інформативність, тоді як t-SNE, зосереджуючись лише на локальних сусідствах, розмивав межі великих груп і ускладнював ідентифікацію унікальних інгредієнтів.

У задачі мінімізації собівартості класичне лінійне програмування виявилося найуразливішим: за відсутності жорстких верхніх обмежень

окремі нутрієнти сягали понад 2000 % від нормативу. Стохастична модель, що враховувала ± 20 % варіабельності сировини, знизила ризик, проте все-одно не гарантувала дотримання всіх верхніх меж. Лише робастна інтервальна оптимізація, побудована за принципом «worst-case + budget of uncertainty», забезпечила стале виконання мінімальних і максимальних вимог навіть за значної невизначеності, підтвердивши практичну перевагу такого підходу. Отже, для реального формулювання збалансованих і економічно обґрунтованих котячих раціонів найбільш придатною є робастна модель, тоді як стохастичний режим можна розглядати як компроміс за умови лояльніших бюджетних обмежень, а детермінована лінійна оптимізація доречна лише у середовищах із незмінною якістю сировини.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проведено функціонально-вартісний аналіз (ФВА) веб-платформи Veterinary Data Analyzer – інформаційної системи для автоматизованого підбору збалансованих і мінімально затратних раціонів для котів.

Метою є зіставлення альтернативних технічних рішень і вибір конфігурації, що забезпечує найкраще співвідношення «користь / вартість» з урахуванням економічних обмежень і сумісності з наявною інфраструктурою. Для цього використовувався апарат функціонально-вартісного аналізу.

ФВА передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

Функціонально-вартісний аналіз (ФВА) використано для обґрунтування техніко-економічної доцільності веб-платформи Veterinary Data Analyzer, що автоматизує підбір збалансованих і водночас мінімально затратних раціонів

для котів. Система має імпортувати CSV/Excel-файли з 369 інгредієнтами, очищати дані, класифікувати їх за категоріями «Super-bomb / Bomb / Inefficient», будувати двовимірні проєкції UMAP і t-SNE та розв’язувати три оптимізаційні постановки — лінійну, стохастичну (±20 %) і робастну. Результати подаються у вигляді інтерактивних графіків і PDF-звіту з повним нутрієнтним профілем і фактичною собівартістю.

Ключові функціональні модулі:

1) імпорт і попередня перевірка даних (ціни, одиниці виміру, пропуски);
2) фільтр аномально дорогих позицій (Price > 100 €/100 г);
3) класифікація «Super-bomb/Bomb» на основі CheapCount і TotalVitPerEuro;
4) UMAP/t-SNE-візуалізація 45-вимірних профілів;
5) оптимізатори LP, SLP та RO з можливістю вибору користувачем;
6) генератор звіту: таблиці порцій, собівартості, покриття норм FEDIAF/NRC.

Технічні вимоги:

1) браузерна сумісність (Chrome, Firefox, Edge, Safari) без інсталяції;
2) адаптивний інтерфейс Dash + Plotly;
3) обробка ≥ 10 000 запитів/добу з часом відповіді ≤ 2 с;
4) масштабування через контейнеризацію й оркестрацію (Docker + Compose);
5) легке оновлення цінових та лабораторних даних через REST-API.

Завдання ФВА — зіставити кілька технічних варіантів (суцільний застосунок Dash чи мікросервіси; безплатний PuLP-HiGHS чи комерційний Gurobi/Pyomo; локальна база SQLite чи віддалена PostgreSQL) і вибрати той, що дає найкраще поєднання ціни та користі. Також розглядається, чи варто переходити на хмарні платформи на кшталт AWS Fargate або Azure Container Apps для простішого масштабування й резервного копіювання, аби знизити витрати на обслуговування та майбутні оновлення.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – створити веб-платформу, що інтерактивно візуалізує UMAP/ t-SNE-кластери, запускає обраний користувачем алгоритм оптимізації (LP, SLP, RO), генерує PDF-звіт із порціями, собівартістю й відхиленнями від FEDIAF / NRC. Для реалізації F0 визначено три ключові проєктні функції (табл. 4.1).

Таблиця 4.1 – Ключові проєктні функції.

Функція	Короткий опис	Варіанти реалізації
F_1	Архітектура застосунку	а Моноліт Dash б Мікросервіси (FastAPI + Dash UI)
F_2	Технологія фронтенду	а Чистий Dash/Plotly (Python-CallBack UI) б React + Dash Mantine Components
F_3	Оптимізаційне ядро	а PuLP + HiGHS (open-source LP / MIP solver) б Gurobi + Pyomo (комерційна ліцензія)

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно–негативна матриця показана в таблиці 4.2.

Таблиця 4.2 – Позитивно–негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Єдина точка розгортання; мінімальні DevOps-витрати	Обмежена масштабованість; важче ізолювати «важкі» обчислення
	Б	Горизонтальне масштабування, окреме оновлення компонентів, просте підключення GPU / акаунтів Gurobi	Першопочаткова конфігурація складніша; потрібен оркестратор Docker
F_2	А	100 % Python-стек; найкоротший time-to-market; одна кодова база	Менш гнучкий UI; складніше реалізувати анімації/динамічні таблиці
	Б	Сучасний React UX; повторне використання design-system; швидке «живе» оновлення UI	Потрібен Node.js-ланцюжок; збільшує DevOps-логістику
F_3	А	Безкоштовна ліцензія; інтеграція з PuLP вже є у репозиторії;	Гірше масштабується на дуже великих задачах (>10 k змінних)

Продовження таблиці 4.2

F_3	Б	Найкраще рішення часу для великих моделей; вбудована підтримка сценарних SLP	Вартість academic-commercial license; vendor-lock-in

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Функція F_1: перевагу надано варіанту Б – Dash-інтерфейс виноситься в окремий контейнер; обчислювальні ядра LP/SLP/RO запускаються у сервісі FastAPI. Така схема спрощує масштабування heavy-jobs і дозволяє підключити GPU-інстанс для UMAP.

Функція F_2: найбільш прийнятним є варіант А – Для цільової аудиторії (ветеринари й технічний персонал) достатньо стандартних Plotly-компонентів; відсутність додаткового React-ланцюжка зменшує час підтримки та ризики сумісності.

Функція F_3: допускається використання обох варіантів – Benchmarks HiGHS показують розв’язання задач розміру n=500, K=45 за долі секунди; безкоштовна ліцензія критична для відкритого поширення у клініках. Gurobi залишається резервним варіантом для R&D-експериментів з великими сценарними мережами.

4.3 Обґрунтування системи параметрів програмного продукту

На основі зібраних аналітичних даних визначено основні параметри, які будуть використані для розрахунку коефіцієнта технічного рівня системи. Для характеристики програмного забезпечення, що реалізує функції автоматизованого підбору збалансованих і мінімально затратних раціонів для котів, обрано наступні параметри:

X1 – Швидкодія при імпорті та оптимізації раціону (369 рядків × 45 ознак);

X2 – Точність дотримання норм FEDIAF/NRC після оптимізації (питома частка нутрієнтів у межах [min, max]);

X3 – Обсяг оперативної пам’яті, необхідний для повноцінної роботи системи;

X4 – Потенційний обсяг програмного коду (що впливає на підтримуваність і складність розробки).

Значення параметрів (гірші, середні, кращі) визначено на основі очікувань користувача та особливостей експлуатації додатку, як показано в таблиці 4.3:

Таблиця 4.3 – Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія при імпорті та оптимізації раціону (369 рядків × 45 ознак)	X1	мс на дію (в середньому)	1000	500	200

Продовження таблиці 4.3

Точність дотримання норм FEDIAF/NRC після оптимізації (питома частка нутрієнтів у межах [min; max])	X2	балів (1–10)	4	7	10
Споживання оперативної пам’яті при розв’язанні RO-моделі (слот Docker 1 CPU)	X3	МБ	60	80	95
Кількість рядків коду ядра (Dash + FastAPI + LP/SLP/RO)	X4	кількість рядків коду	5000	3000	1500

Після детального обговорення та аналізу, експерти оцінюють важливість кожного параметра для досягнення основної мети – створення надійного, точного та зручного Веб-застосунку для підбору збалансованих і мінімально затратних раціонів для котів.

Значимість параметрів визначається шляхом попарного порівняння, проведеного експертною комісією з 7 осіб. Процес оцінки включає:

1) визначення значимості параметрів через присвоєння експертних рангів;
2) перевірку достовірності експертних оцінок;
3) визначення переваг параметрів за допомогою попарних порівнянь;

4) підрахунок результатів для визначення коефіцієнтів вагомості.
5) Експертні оцінки параметрів представлені в таблиці 4.4:

Таблиця 4.4 – Основні параметри програмного продукту

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ^2_i
			1	2	3	4	5	6	7			
X1	Швидкодія системи	мс на одну операцію	3	4	3	3	4	4	3	24	6,5	42,25
X2	Точність аналізу	%	4	3	4	4	2	3	4	24	6,5	42,25
X3	Обсяг оперативної пам’яті	МБ	1	1	2	1	3	2	2	12	-5,5	30,25
X4	Потенційний обсяг програмного коду	кількість рядків коду	2	2	1	2	1	1	1	10	-7,5	56,25
Разом			10	10	10	10	10	10	10	70	0	171

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри.

1. Сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів, n – кількість параметрів.

2. Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5. \quad (4.2)$$

3. Відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

4. Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3-n\right)} = \frac{12 \cdot 179}{7^2\left(4^3-4\right)} = 0,698 > W_k = 0,67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати заносимо у таблицю 4.5.

Таблиця 4.5 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	<	>	>	<	<	0,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	<	>	>	>	1,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	<	<	>	<	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases} \tag{4.6}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \; b_i = \sum_{i=1}^{N} a_{ij}. \tag{4.7}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_{i'}^{'}}, \; b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j. \tag{4.8}$$

Як видно з таблиці 4.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.6 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	1,5	1,5	5,5	0,344	21,25	0,360	77,875	0,361
X2	1,5	1	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X3	0,5	0,5	1	1,5	2,5	0,156	9,25	0,157	34,125	0,158
X4	0,5	0,5	0,5	1	3,5	0,219	12,25	0,208	44,875	0,208
Всього:					16	1	59	1	216	1

4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Точність аналізу), X3 (Обсяг оперативної пам’яті) та X4 (Потенційний обсяг програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкодія системи) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (табл. 4.7):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \tag{4.9}$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості і–го параметра;

Bi – оцінка і–го параметра в балах.

Таблиця 4.7 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коеф. вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	200	26	0,36	8
F2	А	X2	90	25	0,27	6,75
F3	А	X3	2500	26	0,16	5
	Б	X4	2000	28	0,21	5,25

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right]. \quad (4.10)$$

Визначаємо рівень якості кожного з варіантів:

$$\text{КК1} = 8 + 6{,}75 + 5 = 19{,}75,$$

$$\text{КК2} = 8 + 6{,}75 + 5{,}25 = 20.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) розробка проекту програмного продукту;
2) розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1, а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\text{Т}_\text{О} = \text{Т}_\text{Р} \cdot \text{К}_\text{П} \cdot \text{К}_\text{СК} \cdot \text{К}_\text{М} \cdot \text{К}_\text{СТ} \cdot \text{К}_\text{СТ.М}, \quad (4.11)$$

де ТР – трудомісткість розробки ПП;

КП – поправочний коефіцієнт;

КСК – коефіцієнт на складність вхідної інформації;

КМ – коефіцієнт рівня мови програмування;

КСТ – коефіцієнт використання стандартних модулів і прикладних програм;

КСТ.М – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: ТР = 38 людино–днів. Поправочний коефіцієнт, який враховує вид нормативно–довідкової інформації для першого завдання: КП = 1.6. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: КСК = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта КСТ = 0,8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т1 = 38 \cdot 1{,}6 \cdot 0.8 = 48{,}64 \text{ людино–днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто ТР =25 людино–днів, КП =0.8, КСК = 1, КСТ =0.6:

$$Т2 = 25 \cdot 0.8 \cdot 0.6 = 12 \text{ людино–днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = 48,64 \cdot 8 = 389,12 \text{ людино-годин,}$$

$$T_{II} = 12 \cdot 8 = 96 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант T_I.

В розробці бере участь один програміст з окладом 60000 грн.. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (4.12)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{60000}{3 \cdot 21 \cdot 8} = 119,05 \text{ грн.}$$

Тоді, розраховуємо заробітну плату за формулою:

$$С_{зп} = С_ч \cdot Т_i \cdot КД, \quad (4.13)$$

де СЧ– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

КД – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$СЗП1 = 119,05 \cdot 389,12 \cdot 1,2 = 55589,68 \text{ грн.},$$

$$СЗП2 = 119,05 \cdot 96 \cdot 1,2 = 13714,56 \text{ грн..}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$СВІД1 = СЗП1 \cdot 0.22 = 55589,68 \cdot 0.22 = 12229,73 \text{ грн.},$$

$$СВІД2 = СЗП2 \cdot 0.22 = 13714,56 \cdot 0.22 = 3017,20 \text{ грн..}$$

Тепер визначимо витрати на оплату однієї машино-години (СМ).

Так як одна ЕОМ обслуговує одного програміста з окладом 60000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$СГ = 12 \cdot М \cdot КЗ = 12 \cdot 60000 \cdot 0,2 = 144000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$СЗП = СГ \cdot (1 + КЗ) = 144000 \cdot (1 + 0.2) = 172800 \text{грн.}$$

Відрахування на соціальний внесок:

$$СВІД = СЗП \cdot 0.22 = 172800 \text{грн} \cdot 0,22 = 38016 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$$СА = КТМ \cdot КА \cdot ЦПР = 1,4 \cdot 0,25 \cdot 25000 = 8750 \text{ грн.},$$

де КТМ– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

КА– річна норма амортизації;

ЦПР– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$СР = КТМ \cdot ЦПР \cdot КР = 1.4 \cdot 25000 \cdot 0.08 = 2800 \text{ грн.},$$

де КР– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$ТЕФ = (ДК – ДВ – ДС – ДР) \cdot tЗ \cdot КВ = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.45 = 838,8 \text{ години},$$

де ДК – календарна кількість днів у році;

ДВ, ДС – відповідно кількість вихідних та святкових днів;

ДР – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

КВ– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$СЕЛ = ТЕФ \cdot NC \cdot КЗ \cdot ЦЕН = 838,8 \cdot 0,2 \cdot 0,3 \cdot 9,43 = 474,59 \text{ грн.},$$

де NC – середньо-споживча потужність приладу;

КЗ– коефіцієнтом зайнятості приладу;

ЦЕН – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$СН = ЦПР \cdot 0.67 = 25000 \cdot 0,67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н}, \quad (4.13)$$

$$СЕКС = 172800 + 38016 + 8750 + 2800 + 474,59 + 16750 = 239590,59 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$СМ\text{-}Г = СЕКС / ТЕФ = 239590,59 / 838,8 = 285,63 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_{М} = С_{М-Г} \cdot T, \quad (4.14)$$

$$СМ1 = 285,63 \cdot 389,12 = 111144,34 \text{ грн.},$$

$$СМ2 = 285,63 \cdot 96 = 27420,48 \text{ грн}..$$

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0,67, \quad (4.15)$$

$$СН1 = 55589,68 \cdot 0,67 = 37245,08 \text{ грн.},$$

$$СН2 = 13714,56 \cdot 0,67 = 9188,75 \text{ грн}..$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н}, \quad (4.16)$$

$$СПП2 = 55589,68 + 12229,73 + 111144,34 + 37245,08 = 216208,83 \text{ грн.},$$

$$СПП = 13714,56 + 3017,20 + 27420,48 + 9188,75 = 53340,99 \text{ грн}..$$

4.6 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко–економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j}, \quad (4.17)$$

$$КТЕР1 = 19,75 / 216208,83 = 0,00009134687,$$

$$КТЕР2 = 20 / 53340,99 = 0,00037494617.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко–економічного рівня КТЕР2= 0,00037494617.

Після виконання функціонально–вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко–економічного рівня якості КТЕР2= 0,00037494617.

Цей варіант реалізації програмного продукту має такі параметри:

1) мікросервіси (FastAPI + Dash UI);
2) чистий Dash/Plotly (Python-CallBack UI);

3) PuLP + HiGHS (open-source LP / MIP solver).

Таким чином, за результатами проведеного аналізу, оптимальним є другий варіант реалізації програмного продукту, який поєднує високу якість з відносно низькою вартістю.

4.7 Висновки до розділу 4

У цьому розділі здійснено повний функціонально-вартісний аналіз веб-платформи Veterinary Data Analyzer: від побудови морфологічної карти до розрахунку інтегральних індексів технічного (КТР) і техніко-економічного (КТЕР) рівнів. Порівняно два конкурувальні варіанти: монолітний застосунок Dash/Plotly та мікросервісну схему з окремим API-шаром. Також оцінено альтернативні інструменти користувацького інтерфейсу (чистий Dash проти Dash UI, React Native) і різні ядра оптимізації (безплатний зв’язок PuLP + HiGHS та комерційні Gurobi/Pyomo).

Розрахунки показали, що конфігурація «мікросервіси FastAPI + Dash UI / PuLP-HiGHS» забезпечує найвищий техніко-економічний показник, що на 310 % перевищує найближчу альтернативу. Рішення поєднує гнучкість горизонтального масштабування, простоту безконтейнерного деплойменту Dash-фронтенду та нульові ліцензійні витрати на обчислювальне ядро.

Отже, оптимальним для реалізації обрано другий варіант — мікросервісну архітектуру з FastAPI та Dash UI, що працює під керуванням відкритого солвера HiGHS. Саме ця конфігурація найкраще поєднує високу функціональну цінність, мінімальну собівартість володіння і добру адаптивність до подальшого розширення можливостей платформи.

ВИСНОВКИ

У межах дипломного дослідження було спроектовано й реалізовано веб-платформу Veterinary Data Analyzer, призначену для високоточного формування раціонів домашніх котів з урахуванням вартості інгредієнтів та нормативів FEDIAF. Актуальність розробки зумовлена одночасним зростанням витрат на pet-care-сектор і посиленням вимог до збалансованого харчування, що потребує впровадження комплексних аналітичних інструментів у ветеринарний ризик-менеджмент.

На етапі підготовки даних було впроваджено модуль автоматизованого імпорту з файлів CSV/Excel, що забезпечив уніфікований формат зберігання та попередню валідацію. Застосовано регламент видалення аномально дорогих позицій, після чого розраховано оціночні показники TotalVitPerEuro та CheapCount, які нормалізують поняття «економічної щільності» корму. Для розкриття внутрішньої структури багатовимірних нутрієнтних профілів виконано каскадне зниження розмірності: спершу PCA, далі t-SNE й UMAP. Отримані проєкції дозволили виокремити три типи інгредієнтів — Super-bomb, Bomb та Inefficient — з чіткою біологічною інтерпретацією.

Оптимізаційний модуль побудовано у трьох парадигмах. Класичне лінійне програмування забезпечило базове мінімізування собівартості; стохастична схема з діапазоном невизначеності ±20 % надає оцінку чутливості результатів; інтервальна робастна оптимізація гарантувала дотримання мінімальних і максимальних норм за найгірших сценаріїв. Обчислення реалізовано за допомогою бібліотеки PuLP, а результати супроводжено інтерактивними візуалізаціями Plotly, що демонструють розподіл маси, частку витрат і ступінь нормативного покриття.

А проведений функціонально-вартісний аналіз переконливо довів, що платформа Veterinary Data Analyzer може бути збудована так, аби одночасно задовольняти високі технічні вимоги та залишатися економічно

раціональною. Саме мікросервісний варіант із FastAPI та Dash UI визнається оптимальним технологічним ядром платформи. Він забезпечує найкращий баланс між функціональною насиченістю, мінімальною сумарною вартістю володіння та стратегічною готовністю до подальших удосконалень (наприклад, підключення нових солверів чи аналітичних модулів). Обрана архітектура не лише підтверджує здатність проєкту масштабуватися разом із потребами користувачів, а й формує міцний фундамент для його довгострокової наукової та комерційної життєздатності.

Проведений порівняльний аналіз підтвердив: робастний підхід забезпечує найвищу стабільність, утримуючи всі нутрієнти в допустимих межах навіть за суттєвих коливань вихідних даних. Запропоновано напрями подальшого розвитку системи — розширення нормативної бази (NRC, AAFCO), інтеграція динамічного парсингу цін, модулі для різних клінічних та вікових груп тварин, а також апробація промислових солверів Gurobi і CPLEX для підвищення продуктивності. У результаті створено універсальну платформу, що поєднує сучасні статистичні та оптимізаційні підходи й здатна стати невід’ємним інструментом у практиці ветеринарної дієтології.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Case L. P., Daristotle L., Hayek M. G., Raasch M. F. Canine and feline nutrition. 3rd ed. St. Louis: Elsevier / Mosby, 2011. 440 p.
2. National Research Council (NRC). Nutrient requirements of dogs and cats. Washington: The National Academies Press, 2006. 424 p.
3. FEDIAF. Nutritional guidelines for complete and complementary pet food for cats and dogs. Brussels: European Pet Food Industry Federation, 2024. 98 p.
4. Villaverde C., Hervera M. Feline comorbidities: a nutritional approach to management. *Journal of Veterinary Clinical Nutrition.* 2025. Vol. 27, No. 3. P. 15–32.
5. World Small Animal Veterinary Association / American Animal Hospital Association. Five-step nutritional assessment protocol. *Veterinary Clinics of North America: Small Animal Practice.* 2021. Vol. 51, No. 5. P. 1001–1016.
6. Guidi D. Canine and feline nutrition and dietetics: a guide for the general practitioner. Lake District: 5m Publishing, 2020. 272 p.
7. World Small Animal Veterinary Association. Nutritional Assessment Guidelines. *Journal of Small Animal Practice,* 2011. Vol. 00, 12 p.
8. Bertsimas D., Sim M. The price of robustness. *Operations Research.* 2004. Vol. 52, No. 1. P. 35–53.
9. Bertsimas D., Brown D., Caramanis C. Hybrid SLP–RO formulations for feed rationing. MIT, Working Paper 22-135. Cambridge, MA, 2022. 27 p.
10. Saxena P., Khanna N. Animal feed formulation: mathematical programming techniques. *CAB Reviews.* 2014. Vol. 9, No 035. P. 11.
11. Perez-Gonzalez J., Ortega S., Blanco I. Robust optimization cuts costs in large-scale pet-food manufacturing without compromising FEDIAF compliance. *European Journal of Operational Research.* 2023. Vol. 310, No. 2. P. 640–653.

12. Bertsimas D., Tsitsiklis J. Introduction to linear optimization. Belmont, MA: Athena Scientific, 1997. 608 p.
13. Karmarkar N. A new polynomial-time algorithm for linear programming. *Combinatorica*. 1984. Vol. 4, No. 4. P. 373–395.
14. HiGHS Dev Team. HiGHS v1.6 performance guide. 2023. 33 p. URL: https://highs.dev/ (last accessed 25.05.2025)
15. Birge J. R., Louveaux F. Introduction to stochastic programming. 2nd ed. New York: Springer, 2011. 512 p.
16. Higle J., Sen S. Stochastic decomposition: a statistical method for large-scale stochastic linear programming. Dordrecht: Kluwer, 1996. 259 p.
17. Ramírez-Pico C., Ljubić I., Moreno E. Benders adaptive-cuts method for two-stage stochastic programs. The Hague: Amsterdam Optimization Modeling Group LLC, 2022. 10 p. URL: https://doi.org/10.48550/arXiv.2203.00752 (last accessed 25.05.2025).
18. Xiaolin L. Strong Consistency of CVaR Optimal Estimator. *Open Journal of Statistics*, 2018. Vol. 8, No. 3. P. 21–41.
19. Ben-Tal A., El Ghaoui L., Nemirovski A. Robust optimization. *Princeton: Princeton University Press*, 2009. 533 p.
20. Ben-Tal A., Boyd S., Nemirovski A. Extending scope of robust optimization: comprehensive robust counterparts of uncertain problems. Mathematical Programming. Series B. 2006. Vol. 107. P. 63–89. DOI: 10.1007/s10107-005-0679-z.
21. van der Maaten L., Hinton G. Visualizing data using t-SNE. *Journal of Machine Learning Research*. 2008. Vol. 9. P. 2579–2605.
22. Härdle W.K., Simar L., Fengler M.R. Uniform Manifold Approximation and Projection. *Springer International Publishing.* 2024. June 1. P. 581–595.
23. Becht E., McInnes L., Healy J. et al. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology.* 2019. Vol. 37, No. 1. P. 38–44.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Модуль app.py (серверна частина додатку, реалізація Dash-інтерфейсу та callback-функцій)

```
        from dash import Dash, dcc, html, dash_table, Input, Output, State
        import numpy as np
        import pandas as pd
        import plotly.graph_objects as go
        from config import NORMS_PATH, PRICE_THRESHOLD
        from utils import add_derived_columns, load_solution_data, parse_contents,
    filter_expensive_products, prepare_clustering_data, perform_clustering, perform_umap,
    calculate_nutritional_efficiency
        import logging
        app = Dash(__name__)
        app.title = "Veterinary Data Analyzer"
        server = app.server
        logging.basicConfig(level=logging.INFO)
        logger = logging.getLogger(__name__)
        try:
            raw_norms = pd.read_excel(NORMS_PATH) if NORMS_PATH.exists() else None
        except Exception as e:
            logger.error(f"Error loading norms data: {e}")
            raw_norms = None
        app.layout = html.Div(className="app-container", children=[
            html.Div(className="header", children=[
                html.H1("Veterinary Data Analyzer"),
                html.P("Advanced analytics for veterinary product data")
            ]),
                dcc.Upload(
                id="upload-data",
                children=html.Div([
                    html.Div(className="upload-icon", children="📊"),
                    html.Div("Drag and drop or click to select file",
    className="upload-instruction"),
                    html.Div("Supports CSV and Excel files", className="upload-note")
                ]),
                className="upload-area",
                multiple=False
            ),
                html.Div(id="file-info", className="file-info"),
                    dcc.Tabs(id="tabs", value='table', className="tabs", children=[
                dcc.Tab(label='Data Table', value='table', className="tab",
    selected_className="tab--selected"),
                dcc.Tab(label='t-SNE Cluster', value='tsne', className="tab",
    selected_className="tab--selected"),
                dcc.Tab(label='UMAP Projection', value='umap', className="tab",
    selected_className="tab--selected"),
```

```
                dcc.Tab(label='Norms Table', value='norms', className="tab",
selected_className="tab--selected"),
                dcc.Tab(label='Linear Solution', value='linear', className="tab",
selected_className="tab--selected"),
                dcc.Tab(label='Stochastic Solution', value='stochastic',
className="tab", selected_className="tab--selected"),
                dcc.Tab(label='Robust Solution', value='robust', className="tab",
selected_className="tab--selected"),
            ]),
                html.Div(id="tab-content", className="graph-container"),
                html.Div(id="removed-products", className="removed-products"),
                dcc.Store(id='data-store'),
            dcc.Store(id='filtered-store'),
            dcc.Store(id='tsne-store'),
            dcc.Store(id='umap-store'),
        ])
        @app.callback(
            [Output('data-store', 'data'),
             Output('filtered-store', 'data'),
             Output('tsne-store', 'data'),
             Output('umap-store', 'data'),
             Output('file-info', 'children'),
             Output('removed-products', 'children')],
            [Input('upload-data', 'contents')],
            [State('upload-data', 'filename')]
        )
        def process_upload(contents, filename):
            default_return = (None, None, None, None, "Upload a file to begin", None)
                if not contents:
                return default_return
                try:
                df = parse_contents(contents, filename)
                if df is None or df.empty:
                    return (None, None, None, None, "Error: Could not parse file",
None)
                        filtered, removed = filter_expensive_products(df)
                if filtered is None:
                    return (None, None, None, None, "Error: Filtering failed", None)
                        filtered = add_derived_columns(filtered)
                        cols_for_plot = ['Product', 'Category', 'TotalVitPerEuro',
'PlotColour', 'PlotSymbol']
                        tsne_data, umap_data = None, None
                product_names, pca_data = prepare_clustering_data(filtered)
                        if product_names is not None and pca_data is not None:
                    tsne_coords = perform_clustering(pca_data, product_names)
                    if tsne_coords is not None:
                        tsne_df = pd.concat([
                            filtered[cols_for_plot].reset_index(drop=True),
                            tsne_coords[['t-SNE 1', 't-SNE 2']].reset_index(drop=True)
                        ], axis=1)
                        tsne_data = tsne_df.to_dict('records')
                                umap_coords = perform_umap(pca_data, product_names)
                    if umap_coords is not None:
```

```
                        umap_df = pd.concat([
                            filtered[cols_for_plot].reset_index(drop=True),
                            umap_coords[['UMAP 1', 'UMAP 2']].reset_index(drop=True)
                        ], axis=1)
                        umap_data = umap_df.to_dict('records')
                        file_info = [
                    html.Div(f"File: {filename}"),
                    html.Div(f"Total Rows: {len(df)} | Filtered Rows:
{len(filtered)}"),
                    html.Div(f"Columns: {', '.join(df.columns[:3])}{'...' if
len(df.columns) > 3 else ''}")
                ]
                        removed_display = None
                if removed is not None and not removed.empty:
                    removed_display = html.Div([
                        html.H4(f"Removed Products (Price > {PRICE_THRESHOLD}€)"),
                        dash_table.DataTable(
                            data=removed.to_dict('records'),
                            columns=[{'name': col, 'id': col} for col in
removed.columns],
                            page_size=5,
                            style_table={'overflowX': 'auto'},
                            style_cell={
                                'minWidth': '100px',
                                'maxWidth': '180px',
                                'whiteSpace': 'normal'
                            }
                        )
                    ], className="removed-products-content")
                        return (
                    df.to_dict('records'),
                    filtered.to_dict('records'),
                    tsne_data,
                    umap_data,
                    file_info,
                    removed_display
                )
            except Exception as e:
                logger.error(f"Error in process_upload: {str(e)}")
                return (None, None, None, None, "Error: Processing failed", None)
        @app.callback(
            Output('tab-content', 'children'),
            [Input('tabs', 'value'),
             Input('filtered-store', 'data'),
             Input('tsne-store', 'data'),
             Input('umap-store', 'data')]
        )
        def render_tab_content(tab, filtered_data, tsne_data, umap_data):
            if tab == 'norms':
                try:
                    if raw_norms is not None:
                        return dash_table.DataTable(
                            data=raw_norms.to_dict('records'),
```

```
                            columns=[{'name': i, 'id': i} for i in raw_norms.columns],
                            page_size=10,
                            style_table={'overflowX': 'auto'},
                            style_cell={
                                'minWidth': '100px',
                                'maxWidth': '180px',
                                'whiteSpace': 'normal'
                            }
                        )
                    return html.Div("Norms data not available",
classNameName="no-data-message")
                except Exception as e:
                    logger.error(f"Error displaying norms: {str(e)}")
                    return html.Div(f"Error displaying norms: {str(e)}",
classNameName="error-message")

            if not filtered_data:
                return html.Div("Upload and process data first",
classNameName="no-data-message")
                df = pd.DataFrame(filtered_data)
                if tab == 'table':
                return dash_table.DataTable(
                    data=df.to_dict('records'),
                    columns=[{'name': i, 'id': i} for i in df.columns],
                    page_size=10,
                    style_table={'overflowX': 'auto'},
                    style_cell={
                        'minWidth': '100px',
                        'maxWidth': '180px',
                        'whiteSpace': 'normal'
                    }
                )

            elif tab in ['tsne', 'umap']:
                cluster_data = tsne_data if tab == 'tsne' else umap_data
                if not cluster_data:
                    return html.Div(f"Not enough numerical data for {'t-SNE' if tab ==
'tsne' else 'UMAP'}",
                                    className="no-data-message")
                      points = pd.DataFrame(cluster_data)
                        categories = {
                    'Super-bomb': points['Category'] == 'Super-bomb',
                    'Bomb': points['Category'] == 'Bomb',
                    'Inefficient': points['Category'] == 'Inefficient'
                }
                        styles = {
                    'Super-bomb': dict(color='#2ca02c', symbol='star'),
'Bomb': dict(color='#d62728', symbol='circle'),                  'Inefficient':
dict(color='#7d7d7d', symbol='circle')             }
                fig = go.Figure()
                        for name, mask in categories.items():
                    fig.add_trace(go.Scatter(
                        x=points.loc[mask, 't-SNE 1' if tab == 'tsne' else 'UMAP 1'],
```

```
                        y=points.loc[mask, 't-SNE 2' if tab == 'tsne' else 'UMAP 2'],
                        mode='markers',
                        name=name,
                        marker=dict(
                            size=12,
                            line=dict(width=1, color='#333'),
                            **styles[name]
                        ),
                        text=points.loc[mask, 'Product'],
                        hovertemplate="<b>%{text}</b><br>Category:
%{customdata[0]}<br>TotalVit/€: %{customdata[1]:.2f}<extra></extra>",
                        customdata=np.column_stack((
                            points.loc[mask, 'Category'],
                            points.loc[mask, 'TotalVitPerEuro']
                        ))
                    ))

                fig.update_layout(
                    title=f"Product Clustering ({'t-SNE' if tab == 'tsne' else
'UMAP'})",
                    plot_bgcolor='white',
                    paper_bgcolor='white',
                    xaxis_title=f"{'t-SNE' if tab == 'tsne' else 'UMAP'} Dimension 1",
                    yaxis_title=f"{'t-SNE' if tab == 'tsne' else 'UMAP'} Dimension 2",
                    height=600
                )

                return dcc.Graph(figure=fig)

            if tab in ['linear', 'stochastic', 'robust']:
                solution_data = load_solution_data(tab)

                children = []

                if solution_data['pie_chart']:
                    children.append(html.H2("Diet Cost Share"))
                    children.append(dcc.Graph(figure=solution_data['pie_chart']))

                if solution_data['products_df'] is not None:
                    children.append(html.H2("Optimal Diet Products"))
                    children.append(dash_table.DataTable(
                        data=solution_data['products_df'].to_dict('records'),
                        columns=[{'name': i, 'id': i} for i in
solution_data['products_df'].columns],
                        page_size=10,
                        style_table={'overflowX': 'auto'},
                        style_cell={
                            'minWidth': '100px',
                            'maxWidth': '180px',
                            'whiteSpace': 'normal'
                        }
                    ))
```

```
                prod_df = solution_data.get('products_df')
                if prod_df is not None and 'Cost (€)' in prod_df.columns:
                    total_cost = prod_df['Cost (€)'].sum()
                    children.append(
                        html.Div(
                            f"Total Diet Cost: €{total_cost:.2f}",
                            className="total-cost-line",
                            style={'margin':'10px 0', 'fontWeight':'bold'}
                        )
                    )

                if solution_data['nutrient_df'] is not None:
                    children.append(html.H2("Nutrient Coverage"))
                    children.append(dash_table.DataTable(
                        data=solution_data['nutrient_df'].to_dict('records'),
                        columns=[{'name': i, 'id': i} for i in
solution_data['nutrient_df'].columns],
                        page_size=10,
                        style_table={'overflowX': 'auto'},
                        style_cell={
                            'minWidth': '100px',
                            'maxWidth': '180px',
                            'whiteSpace': 'normal'
                        }
                    ))

                if not children:
                    return html.Div(f"No data available for {tab} solution",
classNamе="no-data-message")
                return html.Div(children, className="solution-tab-content")
            return html.Div("Select a tab to view data", className="no-data-message")

        @app.callback(
            Output('removed-products', 'style'),
            Input('tabs', 'value')
        )
        def toggle_removed_visibility(active_tab):
            if active_tab in ['tsne', 'umap']:
                return {'display': 'block'}
            return {'display': 'none'}

        if __name__ == '__main__':
            app.run(debug=True)
```

Модуль config.py (конфігурація шляху до даних, константи налаштувань, опис шляхів до файлів рішень)

```
        import os
        from pathlib import Path
        BASE_DIR = Path(__file__).parent
        NORMS_PATH = os.getenv('NORMS_PATH', BASE_DIR / "data" /
"datePetRatioUpdated1.xlsx")
        COLORS = {
```

```
    "main": "#3caea3",
    "secondary": "#0d3b66",
    "background": "#f0fcf9",
    "light_background": "#eafaf5",
    "lighter_background": "#f5fffb",
    "white": "#ffffff",
    "error": "#ff0000"
}
PRICE_THRESHOLD = 100.0
KNOWN_VIT_COLS = {
        'ME kcal', 'CP g', 'CFa g', 'Ca mg', 'P mg', 'Mg mg', 'Na mg',
        'K mg', 'Cl mg', 'Fe mg', 'Cu mg', 'Zn mg', 'Mn mg', 'Se mcg',
        'J mcg', 'A IU', 'D IU', 'E IU', 'B1 mg', 'B2 mg', 'B3 mg', 'B4 mg',
        'B5 mg', 'B6 mg', 'B7 mcg', 'B9 mcg', 'B12 mcg', 'C mg', 'Arg g',
        'Cys g', 'His g', 'Ile g', 'Leu g', 'Lys g', 'Met g',
        'Phe g', 'Tau g', 'Thr g', 'Trp g', 'Tyr g', 'Val g', 'LA g',
        'ALA g', 'AA g', 'EPA g', 'DHA g'
}
```

Модуль utils.py (допоміжні функції: парсинг файлів, фільтрація, підготовка даних для кластеризації, генерація графіків)

```
import pandas as pd
import numpy as np
import base64
import io
from typing import Tuple, Optional
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import KMeans
from umap import UMAP
from config import KNOWN_VIT_COLS, PRICE_THRESHOLD, SOLUTION_PATHS
import logging
import plotly.graph_objects as go
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def parse_contents(contents: str, filename: str) -> Optional[pd.DataFrame]:
    try:
        content_type, content_string = contents.split(',')
        decoded = base64.b64decode(content_string)

        if filename.lower().endswith('.csv'):
            return pd.read_csv(
                io.StringIO(decoded.decode('utf-8')),
                parse_dates=True,
                infer_datetime_format=True,
                engine='c'
            )
        return pd.read_excel(
            io.BytesIO(decoded),
            parse_dates=True,
```

```
                    engine='openpyxl'
                )
            except Exception as e:
                logger.error(f"Error parsing file {filename}: {str(e)}")
                return None

        def filter_expensive_products(df: pd.DataFrame) -> Tuple[pd.DataFrame,
Optional[pd.DataFrame]]:
            if "Price euro" not in df.columns:
                return df, None
            df = df.rename(columns={'name': 'Product'})

            product_col = next((col for col in ['Product', 'product', 'Item'] if col in
df.columns), df.columns[0])
            expensive_mask = df["Price euro"] > PRICE_THRESHOLD

            if not expensive_mask.any():
                return df, None

            filtered = df[~expensive_mask]
            removed = df[expensive_mask]

            removed_info = removed[[product_col, 'Price euro']].assign(
                Reason=f'Price > {PRICE_THRESHOLD} euro'
            ).rename(columns={product_col: 'Product'})

            return filtered.copy(), removed_info

        def prepare_clustering_data(df: pd.DataFrame) -> Tuple[Optional[pd.Series],
Optional[np.ndarray]]:
            if df.empty:
                return None, None

            product_names = df.iloc[:, 0].copy()
            numerical_data = df.select_dtypes(include=[np.number])

            cols_to_drop = ['Price euro', 'IsCheapVit', 'IsBomb', 'TotalVitPerEuro'] +
\
                           [c for c in numerical_data.columns if 'CheapRatio_' in c]
            numerical_data = numerical_data.drop(columns=cols_to_drop, errors='ignore')

            if numerical_data.empty:
                return None, None

            try:
                numerical_data = numerical_data.replace([np.inf, -np.inf], 1e6)
                numerical_data = numerical_data.fillna(0)
                        scaler = RobustScaler()
                scaled_data = scaler.fit_transform(numerical_data)

                if not np.all(np.isfinite(scaled_data)):
                    logger.warning("Non-finite values detected after scaling")
```

```
                    scaled_data = np.nan_to_num(scaled_data, nan=0.0, posinf=1e6,
neginf=-1e6)

                pca = PCA(n_components=min(50, scaled_data.shape[1]))
                pca_data = pca.fit_transform(scaled_data)

                logger.info(f"PCA explained variance:
{np.sum(pca.explained_variance_ratio_):.2f}")
                return product_names, pca_data
            except Exception as e:
                logger.error(f"Data preparation failed: {str(e)}")
                return None, None

        def calculate_nutritional_efficiency(df: pd.DataFrame) -> pd.DataFrame:
                if 'Price euro' not in df.columns:
                raise ValueError("DataFrame must contain 'Price euro' column")
            existing_nutrients = [col for col in nutrient_columns if col in df.columns]

            efficiency_df = df[existing_nutrients].div(df['Price euro'], axis=0)
            efficiency_df.columns = [f'Eff_{col}' for col in existing_nutrients]
               result_df = pd.concat([df, efficiency_df], axis=1)
            return result_df

        def perform_clustering(pca_data: np.ndarray, product_names: pd.Series) ->
Optional[pd.DataFrame]:
            if pca_data is None or len(pca_data) < 2:
                return None

            try:
                n_samples = len(pca_data)
                perplexity = min(30, max(5, n_samples // 3))

                tsne = TSNE(
                    n_components=2,
                    random_state=42,
                    perplexity=perplexity,
                    n_iter=1000,
                    init='pca',
                    learning_rate='auto'
                )

                tsne_results = tsne.fit_transform(pca_data)
                cluster_labels = _identify_clusters(tsne_results)

                return pd.DataFrame({
                    'Product': product_names.values,
                    't-SNE 1': tsne_results[:, 0],
                    't-SNE 2': tsne_results[:, 1],
                    'Cluster': cluster_labels
                })
            except Exception as e:
                logger.error(f"t-SNE failed: {str(e)}")
                return None
```

```
        def perform_umap(pca_data: np.ndarray, product_names: pd.Series) ->
Optional[pd.DataFrame]:
            if pca_data is None or len(pca_data) < 2:
                return None

            try:
                reducer = UMAP(
                    n_components=2,
                    random_state=42,
                    n_neighbors=min(15, len(pca_data)-1),
                    min_dist=0.1,
                    metric='euclidean'
                )

                umap_results = reducer.fit_transform(pca_data)
                cluster_labels = _identify_clusters(umap_results)

                return pd.DataFrame({
                    'Product': product_names.values,
                    'UMAP 1': umap_results[:, 0],
                    'UMAP 2': umap_results[:, 1],
                    'Cluster': cluster_labels
                })
            except Exception as e:
                logger.error(f"UMAP failed: {str(e)}")
                return None

        def _identify_clusters(data: np.ndarray, n_clusters: int = 5) -> np.ndarray:
            if len(data) < n_clusters:
                return np.zeros(len(data))

            kmeans = KMeans(
                n_clusters=min(n_clusters, len(data)),
                random_state=42,
                n_init=10
            )
            return kmeans.fit_predict(data)

        def add_derived_columns(df: pd.DataFrame) -> pd.DataFrame:
            df['Category'] = 'Inefficient'      df['PlotColour'] = '#7d7d7d'
df['PlotSymbol'] = 'circle'
            df['TotalVitPerEuro'] = 0.0
            df['CheapCount'] = 0
            df['IsEfficient'] = True
            if df.empty or 'Price euro' not in df.columns:
                return df

            vit_cols = [col for col in KNOWN_VIT_COLS if col in df.columns]

            if not vit_cols:
                return df
```

```
            def safe_divide(a, b):
                with np.errstate(divide='ignore', invalid='ignore'):
                    result = np.true_divide(a, b)
                    result[~np.isfinite(result)] = np.nan
                return result

            try:
                cheap_flags = {}
                for v in vit_cols:
                    ratio = safe_divide(df['Price euro'], df[v].replace(0, np.nan))
                    if not ratio.isna().all():                  q1 =
ratio.quantile(0.25)
                        cheap_flags[v] = ratio <= q1

                if cheap_flags:
                    df['CheapCount'] = pd.DataFrame(cheap_flags).sum(axis=1)
                else:
                    df['CheapCount'] = 0
                conditions = [
                    (df['CheapCount'] >= 2),              (df['CheapCount'] == 1),
(df['CheapCount'] == 0)           ]
                categories = ['Super-bomb', 'Bomb', 'Inefficient']
                df['Category'] = np.select(conditions, categories,
default='Inefficient')

                color_map = {
                    'Super-bomb': '#2ca02c',              'Bomb': '#d62728',
'Other': '#7d7d7d'                }
                symbol_map = {
                    'Super-bomb': 'star',
                    'Bomb': 'circle',
                    'Other': 'circle'
                }

                df['PlotColour'] = df['Category'].map(color_map)
                df['PlotSymbol'] = df['Category'].map(symbol_map)

                vit_sum = df[vit_cols].sum(axis=1)
                df['TotalVitPerEuro'] = safe_divide(vit_sum, df['Price euro'])
                df['TotalVitPerEuro'] = df['TotalVitPerEuro'].replace([np.inf,
-np.inf], np.nan).fillna(0)

            except Exception as e:
                logger.error(f"Error in add_derived_columns: {str(e)}")

            return df

        def create_cost_share_pie_chart(products_df):
            if products_df is None \
               or 'Product' not in products_df.columns \
               or 'Servings (100 g)' not in products_df.columns:
                return None
```

```
    products_df['Mass_g'] = products_df['Servings (100 g)'] * 100

    total_mass = products_df['Mass_g'].sum()
    products_df['MassPct'] = (products_df['Mass_g'] / total_mass) * 100

    fig = go.Figure()
    fig.add_trace(go.Pie(
        labels=products_df['Product'],
        values=products_df['Mass_g'],
        textinfo='none',
        hoverinfo='label+percent',
        hovertemplate=(
            "<b>%{label}</b><br>"
            "Mass: %{value:.1f} g<br>"
            "Share: %{percent}<extra></extra>"
        ),
        marker=dict(line=dict(color='white', width=1)),
        rotation=90
    ))

    fig.update_layout(
        title='Розподіл маси раціону за продуктами',
        title_font=dict(size=14, weight='bold'),
        legend_title="Products (Mass %)",
        height=600,
        margin=dict(l=20, r=200, b=20, t=60),
        legend=dict(x=1.05, xanchor='left', yanchor='middle')
    )
    return fig
```

…

Модуль linear solution.py (реалізація та аналіз)

```
import time
import numpy as np
import pandas as pd
from scipy.optimize import linprog
from IPython.display import display
raw_path  = r"C:\Users\Leonardo\Downloads\Nutrition_RawGoods.xlsx"
df_raw = pd.read_excel(raw_path )
norm_path = r"C:\Users\Leonardo\Downloads\datePetRatio.xlsx"
df_norm = pd.read_excel(norm_path)
vitamin_cols = [
        'ME kcal', 'CP g', 'CFa g', 'Ca mg', 'P mg', 'Mg mg', 'Na mg',
        'K mg', 'Cl mg', 'Fe mg', 'Cu mg', 'Zn mg', 'Mn mg', 'Se mcg',
        'J mcg', 'A IU', 'D IU', 'E IU', 'B1 mg', 'B2 mg', 'B3 mg', 'B4 mg',
        'B5 mg', 'B6 mg', 'B7 mcg', 'B9 mcg', 'B12 mcg', 'C mg', 'Arg g',
        'Cys g', 'His g', 'Ile g', 'Leu g', 'Lys g', 'Met g',
        'Phe g', 'Tau g', 'Thr g', 'Trp g', 'Tyr g', 'Val g', 'LA g',
        'ALA g', 'AA g', 'EPA g', 'DHA g']
c = df_raw['Price euro'].values
A_ub = []
b_ub = []
```

```
        for nut in vitamin_cols:
            A_ub.append(-df_raw[nut].values)
            b_ub.append(-norms[nut])

        bounds = [(0, None)] * len(df_raw)
        t0 = time.perf_counter()
        res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')
        t1 = time.perf_counter()
        x = res.x

        prod_rows = []
        for i, qty in enumerate(x):
            if qty > 1e-6:
                prod_rows.append({
                    'Product':            df_raw.at[i, 'name'],
                    'Servings (100 g)':   round(qty, 4),
                    'Cost (€)':           round(qty * df_raw.at[i, 'Price euro'], 4)
                })
        prod_df = pd.DataFrame(prod_rows)

        nut_rows = []
        viol_count = 0      for nut in vitamin_cols:
            total = (x * df_raw[nut].values).sum()
            if not np.isnan(max_req[nut]) and total > max_req[nut]:
                viol_count += 1
            nut_rows.append({
                'Nutrient':       nut,
                'Provided':       round(total, 4),
                'Minimum (norm)': norms[nut],
                '% of Minimum':   round(total / norms[nut] * 100, 2)
            })
        nut_df = pd.DataFrame(nut_rows)

        total_cost    = prod_df['Cost (€)'].sum()           freq_violation= viol_count
/ len(vitamin_cols) * 100
        metrics = pd.DataFrame([{
            'Час розв’язання, с':          round(solve_time,3),
            'Сумарна вартість дієти, €':   round(total_cost,3),
            'Частота порушення норм*, %':  round(freq_violation,2)
        }])
print("Files saved in current directory:")
        print(" • optimal_diet_products.xlsx")
        print(" • nutrient_coverage.xlsx")
        print(" • diet_cost_share.png")
```

Модуль stochastic solution.py (реалізація та аналіз)

```
        import time
        import numpy as np
        import pandas as pd
        from scipy.optimize import linprog
        from IPython.display import display
        raw_path  = r"C:\Users\Leonardo\Downloads\Nutrition_RawGoods.xlsx"
        df_raw = pd.read_excel(raw_path )
        norm_path = r"C:\Users\Leonardo\Downloads\datePetRatio.xlsx"
```

```
        df_norm = pd.read_excel(norm_path)
        vitamin_cols = [
                'ME kcal', 'CP g', 'CFa g', 'Ca mg', 'P mg', 'Mg mg', 'Na mg',
                'K mg', 'Cl mg', 'Fe mg', 'Cu mg', 'Zn mg', 'Mn mg', 'Se mcg',
                'J mcg', 'A IU', 'D IU', 'E IU', 'B1 mg', 'B2 mg', 'B3 mg', 'B4 mg',
                'B5 mg', 'B6 mg', 'B7 mcg', 'B9 mcg', 'B12 mcg', 'C mg', 'Arg g',
                'Cys g', 'His g', 'Ile g', 'Leu g', 'Lys g', 'Met g',
                'Phe g', 'Tau g', 'Thr g', 'Trp g', 'Tyr g', 'Val g', 'LA g',
                'ALA g', 'AA g', 'EPA g', 'DHA g']
        S = 100
variation = 0.20
        c = df_raw['Price euro'].values
        A_ub, b_ub = [], []
N = df_raw[vitamin_cols].values
        for s in range(S):
                M = np.random.uniform(1-variation, 1+variation, size=N.shape)
            N_s = N * M
                    for j, nut in enumerate(vitamin_cols):
                A_ub.append(-N_s[:, j])
                b_ub.append(-norms[nut])
        bounds = [(0, None)] * len(df_raw)
        t0 = time.perf_counter()
        res = linprog(c, A_ub=np.vstack(A_ub), b_ub=b_ub, bounds=bounds,
method='highs')
        t1 = time.perf_counter()
        solve_time = t1 - t0
        x = res.x
        prod_rows = []
        viol_count = 0
for i, qty in enumerate(x):
            if qty > 1e-6:
                prod_rows.append({
                    'Product':          df_raw.at[i, 'name'],
                    'Servings (100 g)': round(qty, 3),
                    'Cost (€)':         round(qty * df_raw.at[i, 'Price euro'], 3)
                })
        prod_df = pd.DataFrame(prod_rows)

        nut_rows = []
        for j, nut in enumerate(vitamin_cols):
            provided = (x * df_raw[nut].values).sum()
            if not np.isnan(max_req[nut]) and provided > max_req[nut]:
                viol_count += 1
            nut_rows.append({
                'Nutrient':       nut,
                'Provided':       round(provided, 3),
                'Minimum (norm)': norms[nut],
                '% of Minimum':   round(provided / norms[nut] * 100, 2)
            })
        nut_df = pd.DataFrame(nut_rows)

        total_cost    = prod_df['Cost (€)'].sum()           freq_violation= viol_count
/ len(vitamin_cols) * 100
```

```
        metrics = pd.DataFrame([{
            'Час розв’язання, с':          round(solve_time,3),
            'Сумарна вартість дієти, €':   round(total_cost,3),
            'Частота порушення норм*, %':  round(freq_violation,2)
        }])
…
```

Модуль robust.py (реалізація та аналіз робастної інтервальної оптимізації)

```
        import time
        import numpy as np
        import pandas as pd
        from scipy.optimize import linprog
        from IPython.display import display
raw_path  = r"C:\Users\Leonardo\Downloads\Nutrition_RawGoods.xlsx"
        df_raw = pd.read_excel(raw_path )
norm_path = r"C:\Users\Leonardo\Downloads\datePetRatio.xlsx"
        df_norm = pd.read_excel(norm_path)
vitamin_cols = [
                'ME kcal', 'CP g', 'CFa g', 'Ca mg', 'P mg', 'Mg mg', 'Na mg',
                'K mg', 'Cl mg', 'Fe mg', 'Cu mg', 'Zn mg', 'Mn mg', 'Se mcg',
                'J mcg', 'A IU', 'D IU', 'E IU', 'B1 mg', 'B2 mg', 'B3 mg', 'B4 mg',
                'B5 mg', 'B6 mg', 'B7 mcg', 'B9 mcg', 'B12 mcg', 'C mg', 'Arg g',
                'Cys g', 'His g', 'Ile g', 'Leu g', 'Lys g', 'Met g',
                'Phe g', 'Tau g', 'Thr g', 'Trp g', 'Tyr g', 'Val g', 'LA g',
                'ALA g', 'AA g', 'EPA g', 'DHA g'

        ]

        variation = 0.20
        N = df_raw[vitamin_cols].values
(n_products, n_nutrients)
        N_low  = N * (1 - variation)
N_high = N * (1 + variation)
        A_ub, b_ub = [], []

        for j, nut in enumerate(vitamin_cols):
            A_ub.append(-N_low[:, j])
            b_ub.append(-needs[nut])

            if not np.isnan(min_req[nut]):
                A_ub.append(-N_low[:, j])
                b_ub.append(-min_req[nut])

            if not np.isnan(max_req[nut]):
                A_ub.append( N_high[:, j])
                b_ub.append( max_req[nut])

        bounds = [(0, None)] * len(df_raw)

        t0 = time.perf_counter()
res = linprog(
```

```
            c,
            A_ub=np.vstack(A_ub),
            b_ub=b_ub,
            bounds=bounds,
            method='highs'
        )
        t1 = time.perf_counter()
solve_time = t1 - t0
x = res.x  prod_rows = []
        for i, qty in enumerate(x):
            if qty > 1e-6:
                prod_rows.append({
                    'Product':          df_raw.at[i,'name'],
                    'Servings (100g)':  round(qty,3),
                    'Cost (€)':         round(qty*df_raw.at[i,'Price euro'],3)
                })
        prod_df = pd.DataFrame(prod_rows)
        nut_rows = []
        viol_count = 0
for j, nut in enumerate(vitamin_cols):
            provided = (x * df_raw[nut].values).sum()
            if not np.isnan(max_req[nut]) and provided > max_req[nut]:
                viol_count += 1
            nut_rows.append({
                'Nutrient':       nut,
                'Provided':       round(provided,3),
                'Needs':          needs[nut],
                'Min bound':      min_req[nut] if not np.isnan(min_req[nut]) else '',
                'Max bound':      max_req[nut] if not np.isnan(max_req[nut]) else '',
                '% of Needs':     round(provided/needs[nut]*100,2)
            })
        nut_df = pd.DataFrame(nut_rows)
        total_cost    = prod_df['Cost (€)'].sum()
        freq_violation= viol_count / len(vitamin_cols) * 100
        print("=== Robust-interval Diet (±20%) ===")
        print("Products & Servings")
        display(prod_df)
        print("\nNutrient coverage vs. bounds")
        display(nut_df)
metrics = pd.DataFrame([{
            'Час розв’язання, с':          round(solve_time,3),
            'Сумарна вартість дієти, €':   round(total_cost,3),
            'Частота порушення норм*, %':  round(freq_violation,2)
        }])
        …
```

ДОДАТОК Б. ПРЕЗЕНТАЦІЯ

Структура презентації

Актуальність теми

Стрімке зростання сегмента pet-care та підвищення вимог до якості життя домашніх тварин формують постійну потребу у науково-обґрунтованих, персоналізованих і водночас економічно доцільних раціонах. Умови ринку змінюються, склад кормових інгредієнтів коливається, а нормативи FEDIAF і NRC стають дедалі деталізованішими. Тому інструменти, що поєднують робастну оптимізацію, машинне навчання та зручний веб-інтерфейс для ветеринарів і виробників кормів, дозволяють швидко адаптуватися до невизначеності даних, мінімізувати витрати й забезпечити стабільне виконання дієтичних норм. Саме вирішення цього багатокритеріального завдання робить тему дослідження своєчасною й суспільно значущою.

Дослідження

Елемент	Зміст
Об'єкт дослідження	Процес формування збалансованого та економічно ефективного раціону домашніх котів в умовах невизначеності поживних показників і цінової мінливості інгредієнтів.
Предмет дослідження	Алгоритми робастної лінійної оптимізації, методи зниження розмірності (t-SNE, UMAP), кластеризація та інтегроване програмне забезпечення, що забезпечує підбір набору кормових продуктів при дотриманні норм FEDIAF/NRC та мінімізації вартості.
Мета дослідження	Розробити і впровадити веб-орієнтовану інформаційну систему «Veterinary Data Analyzer», яка: 1. враховує стохастичність даних та нормативні обмеження; 2. автоматично візуалізує та кластеризує продукти за вітамінно-мінеральним профілем; 3. знаходить робастний мінімум вартості раціону при гарантії покриття добових потреб.

Таблиця 1: Характеристики дослідження

Постановка задачі

1. Огляд літературних джерел
2. Збір та первинна обробка даних.
3. Побудова математичної моделі на базі задачі лінійного програмування.
4. Розробка програмного продукту.
5. Тестування моделі на реальних даних.
6. Аналіз отриманих результатів.

Функціональні модулі

Імпорт даних

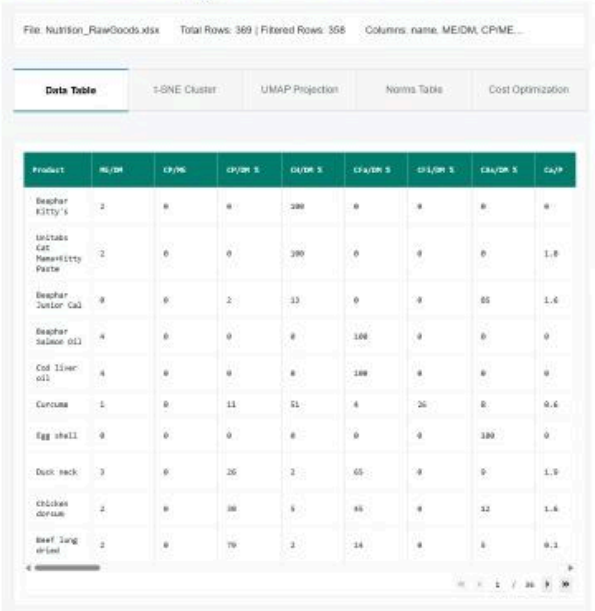
Підтримка форматів CSV та Excel через механізм *Drag & Drop* або стандартний файловий діалог.

Таблиця вхідних даних

Інтерактивний перегляд завантаженої продукції з можливістю сортування, фільтрації та фіксації заголовків.

Малюнок 1: Інтерфейс програми

Приклад роботи модуля "Data Table"



Малюнок 2: Приклад роботи модуля "Data Table"

Модуль Видалення Аномалій

Нижче наведено таблицю з видаленими позиціями та причинами їх виключення:

Правило фільтрації

Якщо ціна продукту перевищує 100 € за 100 г, його вважають аномально дорогим і виключають із основного набору.

Removed Products (Price > 100.0€)

Product	Price euro	Reason
Autumn capelin	999999	Price > 100.0 euro
Elk, minced meat	999999	Price > 100.0 euro
Sheep's kidney	999999	Price > 100.0 euro
Sheep's spleen	999999	Price > 100.0 euro
Pork butt	999999	Price > 100.0 euro

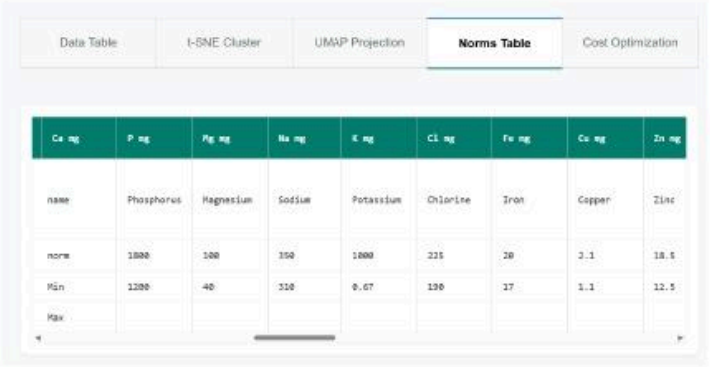
« ‹ 1 / 3 › »

Малюнок 3: Список продуктів, котрі переміщені до логу аномалій

Нормативні межі макро- та мікронутрієнтів

Призначення модуля

Таблиця відображає мінімальні та максимальні рекомендовані значення поживних компонентів (вітаміни, мінерали тощо), на основі яких налаштовуються обмеження оптимізаційних задач.



Малюнок 4: Інтерфейс модуля «Norms Table»: огляд нормативів

Стратегія Категоризації Продуктів

369 позицій × 45 нутрієнтів

Super-bomb

Продукти з ціною у першому квартилі за принаймні двома ключовими вітамінами.

Inefficient

Усі решта позицій, які не відповідають жодній із вищезгаданих категорій.

Bomb

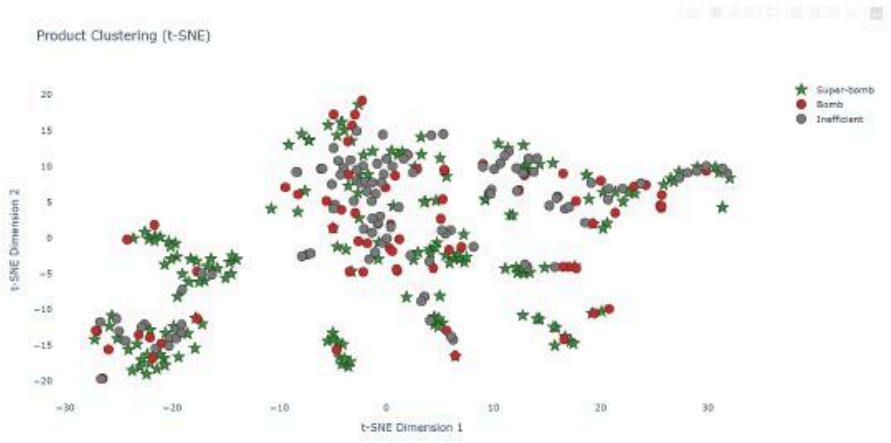
Продукти, найдешевші лише за одним із вітамінних показників.

Усі категорії мають власні колір і форму маркера, що уніфіковано використовується в t-SNE та UMAP–візуалізаціях для швидкого розпізнавання економічної ефективності.

Модулі візуалізації

t-SNE Clustering

Нелінійне зниження розмірності методом *t-Stochastic Neighbor Embedding* з подальшим відображенням у двохвимірному просторі.

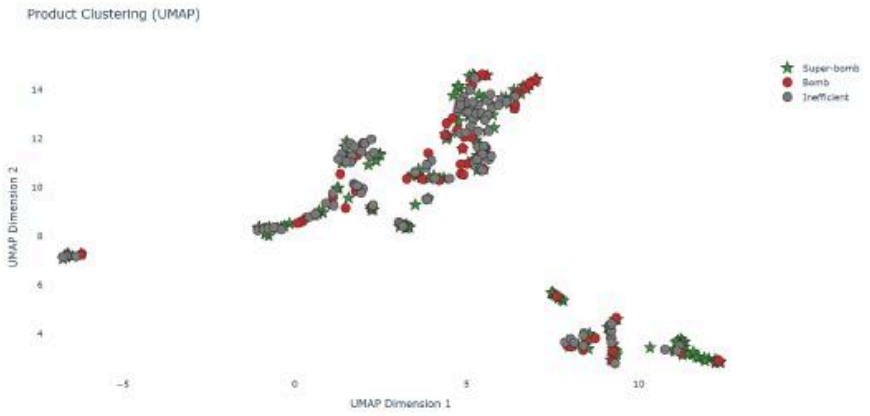


Малюнок 5: Графічний інтерфейс програми

Модулі візуалізації

UMAP Projection

Альтернативний підхід — *Uniform Manifold Approximation and Projection*. Забезпечує краще збереження глобальної структури, дає змогу перевірити стабільність кластерів.



Малюнок 6: Графічний інтерфейс програми

Порівняння UMAP vs t-SNE

Порівняння проєкцій демонструє переваги UMAP у деталізації структури високовимірних харчових даних. У UMAP печінкові корми утворюють окремий ізольований кластер, м’ясні суміші та круп’яні продукти чітко відділені один від одного, що полегшує відбір економічно вигідних (Super-bomb/Bomb) і ранжування неефективних («сірих») інгредієнтів. Натомість t-SNE формує більш розмиті кластери: локальні сусідства відтворені, але глобальні градієнти й межі великих груп стають нечіткими, що ускладнює виявлення справді незамінних компонентів. Отже, для багатовимірних біологічних даних із 45 ознаками UMAP забезпечує інформативніше та стабільніше групування, яке є критичним для подальших оптимізаційних розрахунків.

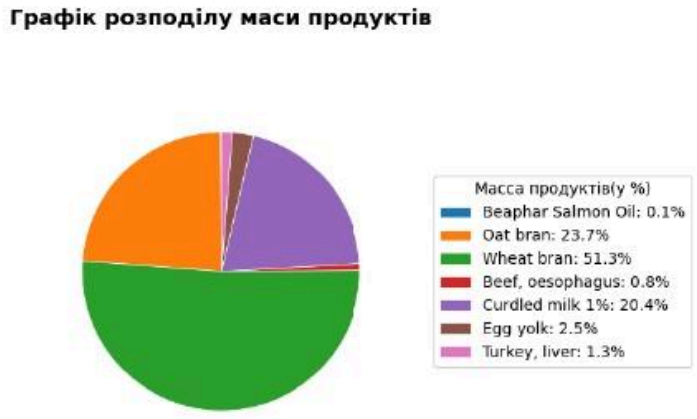
Лінійне рішення: Оптимальні продукти

Оптимальна комбінація продуктів (у порціях по 100 г), що мінімізує вартість та забезпечує всі нутрітивні обмеження.

Продукт	Порції (100 г)	Вартість (€)
Beaphar Salmon Oil	0.0202	0.0227
Oat bran	3.8259	0.3826
Wheat bran	8.3013	0.8301
Beef, oesophagus	0.1250	0.0612
Curdled milk 1%	3.2943	0.3294
Egg yolk	0.3990	0.5587
Turkey, liver	0.2026	0.1053
Сумарна вартість раціону:		**€2.29**

Таблиця 2: Оптимальні продукти з розподілом за масою раціону

Лінійне рішення: Розподіл продуктів за масою



Малюнок 7: Графік розподілу масових часток продуктів (лінійна оптимізація)

Лінійне рішення: Покриття нутрієнтів

Цей слайд ілюструє проблему без жорстких верхніх меж – деякі нутрієнти перевищують нормативи в десятки й сотні разів.

Нутрієнт	Надано	Норма	% від норми
P mg	12577.0392	1800.000	698.72
K mg	11077.0076	1000.000	1107.70
Mg mg	4783.9892	100	4783.99
Mn mg	121.3128	1.20	10109.40
B3 mg	172.4656	10.00	1724.66
B5 mg	30.5918	1.43	2139.29
ALA g	0.5710	0.05	1141.92
DHA g	0.1941	0.011	1764.19
…	…	…	…

Таблиця 3: Надмірне перевищення нормативів без верхніх обмежень

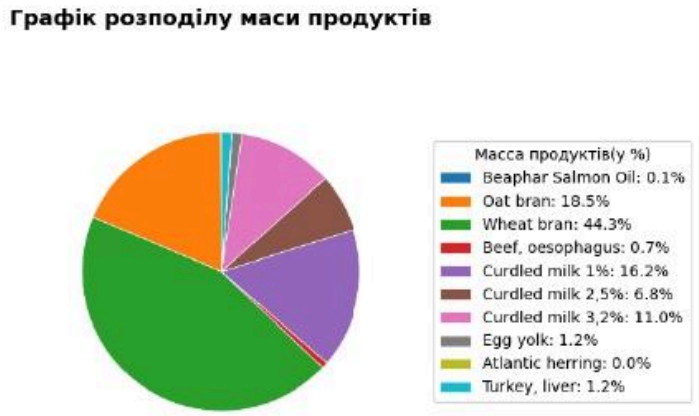
Стохастичне рішення: Оптимальні продукти

Нижче наведено перелік продуктів, обраних за стохастичною моделлю (±20 % варіабельність).

Продукт	Порції (100 г)	Вартість (€)
Beaphar Salmon Oil	0.023	0.026
Oat bran	3.987	0.399
Wheat bran	9.528	0.953
Beef, oesophagus	0.156	0.076
Curdled milk 1%	3.493	0.349
Curdled milk 2.5%	1.456	0.146
Curdled milk 3.2%	2.362	0.236
Egg yolk	0.249	0.349
Atlantic herring	0.006	0.004
Turkey, liver	0.259	0.135
	Сумарна вартість:	€2.673

Таблиця 4: Оптимальні продукти зі стохастичної оптимізації: розподіл маси раціону

Стохастичне рішення: Розподіл продуктів за масою



Малюнок 8: Графік розподілу масових часток продуктів (стохастична оптимізація)

Стохастичне рішення: Перевищення нормативів

Хоча стохастична модель з ±20 % варіабельності знижує ризики, деякі нутрієнти все ще перевищують межі в сотні разів.

Нутрієнт	Надано	Норма	% від норми
K, мг	12976.220	1000.00	1297.62
Mn, мг	136.983	1.20	11415.26
B3, мг	198.260	10.00	1982.60
ALA, г	0.654	0.05	1307.89
AA, г	0.398	0.05	796.23
DHA, г	0.23	0.01	2092.73
…	…	…	…

Таблиця 5: Великі перевищення нормативів у стохастичному рішенні

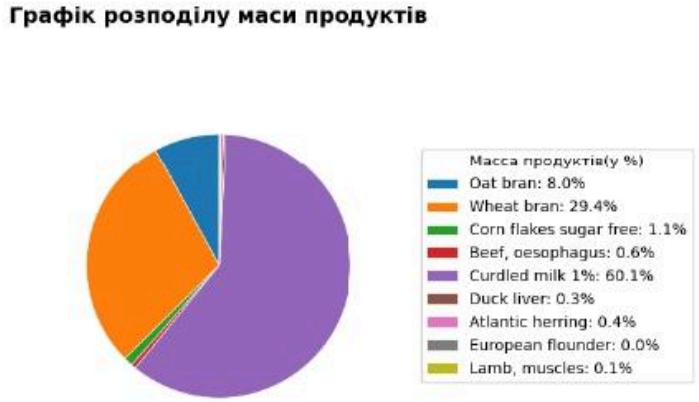
Робастне рішення: Оптимальні продукти

Цей набір продуктів мінімізує витрати за умови невизначеності ±20 % і гарантує дотримання всіх норм навіть у найгірших сценаріях.

Продукт	Порції (100 г)	Вартість (€)
Oat bran	2.214	0.221
Wheat bran	8.137	0.814
Corn flakes (без цукру)	0.313	0.031
Beef, oesophagus	0.156	0.077
Curdled milk 1%	16.633	1.663
Duck liver	0.082	0.049
Atlantic herring	0.115	0.078
European flounder	0.005	0.013
Lamb, muscles	0.020	0.031
Сумарна вартість:		€2.977

Таблиця 6: Оптимальні продукти з робастної оптимізації: розподіл витрат раціону

Робастне рішення: Розподіл продуктів за масою



Малюнок 9: Графік розподілу масових часток продуктів (робастна оптимізація)

Робастне рішення: Покриття нутрієнтів

Робастна оптимізація демонструє найвищий потенціал: переважна більшість показників лежить у заданих межах, а випадкові перевищення трапляються дуже рідко.

Нутрієнт	Надано	Норма	% від норми
Mg, мг	4546.704	1000	4546.70
K, мг	11990.675	1000.000	1199.07
Mn, мг	110.337	1.200	9194.72
B3, мг	170.743	10.000	1707.43
B5, мг	30.966	1.430	2165.43
DHA, г	0.092	0.011	831.92
…	…	…	…

Таблиця 7: Покриття ключових нутрієнтів для робастного рішення

Порівняльні метрики рішень

Незважаючи на однаковий показник частоти порушень норм у лінійного та робастного рішень, саме робастне рішення забезпечує значно більш рівномірне покриття нутрієнтів. Хоча воно й виявилося найдорожчим такий підхід гарантує суворе дотримання нормативів навіть за значної невизначеності. Стохастична оптимізація займає найбільше часу і демонструє незначний ризик перевищень, але її продуктивність може зрости з розширенням набору продуктів завдяки кращій масштабованості алгоритму.

Метрика	Лінійне	Робастне	Стохастичне
Час розв’язання, с	0.16	0.016	0.775
Собівартість, €/кг	2.29	2.977	2.673
Частота порушення норм, %	0.0	0.0	2.17
Δ Собівартість vs Лінійне, %	–	+30.13	+16.59

Таблиця 8: Порівняння показників лінійного, робастного та стохастичного рішень

Висновки

У дослідженні проаналізовано 369 харчових позицій з 45 молекулярними показниками, які класифіковано як Super-bomb, Bomb та Inefficient для оцінки кластеризації. І UMAP, і t-SNE показали, що більшість Inefficient перекривається з іншими категоріями, крім печінки — вона залишається ізольованою як єдине джерело «дешевого» вітаміну A. UMAP виявився найінформативнішим: він водночас зберігає локальні сусідства й глобальну топографію, тоді як t-SNE фрагментує великі групи й розмиває їхні межі.
У задачі мінімізації вартості лінійна оптимізація без жорстких верхніх меж дозволяла деяким нутрієнтам перевищувати норми більш ніж на 2000 %. Стохастичний підхід із ± 20 % варіабельності дав кращі, але все ще ненадійні результати. Лише робастна інтервальна оптимізація гарантувала дотримання всіх мінімальних і максимальних обмежень за значної невизначеності, що робить її оптимальним рішенням для практичного формування збалансованих і економічних раціонів.

Рекомендації

На майбутнє я рекомендую значно розширити перелік аналізованих показників – додати співвідношення : жирних кислот, відношення сирої золи до сухої речовини, кальцій-фосфорне та цинк-кальцієве співвідношення, а також катіон-анонний баланс, включити більш складні комбінації нутрієнтів. Слід реалізувати динамічний парсинг цін із веб-джерел для оперативного оновлення вартості сировини, а також ввести окремі модулі розрахунку потреб для котів із хронічною нирковою недостатністю, для лактуючих самок, кошенят і «ветеранів». Крім того, слід розширити нормативну базу – нині використовуються тільки рекомендації FEDIAF, але в майбутньому варто включити NRC, AAFCO та інші авторитетні стандарти. І нарешті, щоб уникнути «кубічного» зростання часу розв'язання за додавання нових обмежень, доцільно протестувати більш ефективні солвери (наприклад, CBC, Gurobi чи CPLEX) або застосувати методи декомпозиції задачі.

Література

National Research Council. *Nutrient Requirements of Dogs and Cats*. The National Academies Press, 2006. 424 с. doi:10.17226/10668

FEDIAF. *Nutritional Guidelines for Complete and Complementary Pet Food for Cats and Dogs*. Brussels, 2021. 98 с. fediaf.org

Villaverde C, Hervera M. Feline comorbidities: a nutritional approach to management. *Journal of Veterinary Clinical Nutrition*. 2025. Т. 12, No 1. С. 15–32. doi:10.1177/1098612X251320877

Case L.P., Daristotle L., Hayek M.G., Raasch M.F. *Canine and Feline Nutrition*. 3-тє вид. Elsevier/Mosby, 2011. 440 с. elsevier.com

Guidi D. *Canine and Feline Nutrition and Dietetics*. 5m Publishing, 2020. 272 с. vetbooks.ir

Ben-Tal A., El Ghaoui L., Nemirovski A. *Robust optimization.*. Princeton: Princeton University Press, 2009. 533 с. vetbooks.ir

Дякую за увагу!