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: «Підтримка особистої ефективності за допомогою застосунка для ведення нотаток з автоматичною категоризацією та аналізом продуктивності»

Виконав:
студент IV курсу, групи КА-12
Терещук Андрій Володимирович _____

Керівник:
Асистент каф. ММСА Древаль Максим Михайлович _____

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

Консультант з нормоконтролю:
Асистент каф. ММСА Канцедал Георгій Олегович _____

Рецензент:
Доцент каф. СП, к.т.н., Безносик Олександр Юрійович _____

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

Київ – 2025 рік

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

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 студенту

Терещуку Андрію Володимировичу

1. Тема роботи «Підтримка особистої ефективності за допомогою застосунка для ведення нотаток з автоматичною категоризацією та аналізом продуктивності», керівник роботи Древаль Максим Михайлович, асистент, затверджені наказом по університету від «26» травня 2025 р. №1759-с

2. Термін подання студентом роботи: 12.06.2025

3. Вихідні дані до роботи:

   Пуста база даних нотаток.

   Іконка програми інтерфейсу користувача.

4. Зміст роботи:

   1 – Огляд історичних та сучасних методів і рішень для нотування та впорядкування персональних задач; 2 – Теоретичні основи автоматизованої категоризації нотаток й аналізу персональної продуктивності, програмної реалізації; 3 – Реалізація додатку для нотаток з автоматичною категоризацією

та аналізом продуктивності; 4 – Функціонально-вартісний аналіз програмного забезпечення.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

   Зображення існуючих рішень, графічний опис архітектури застосунку, ілюстрації роботи програмного продукту, презентація

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: 14.04.2025

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та формулювання завдання	31.03.2025-06.04.2025	виконано
2	Вивчення літератури та обробка даних за темою роботи	07.03.2025-13.04.2025	виконано
3	Підготовка математичних моделей, розробка програмного коду їх реалізацій	14.04.2025-18.05.2025	виконано
4	Робота над розділами під час переддипломної практики	14.04.2025-18.05.2025	виконано
5	Оформлення та узгодження пояснювальної записки і розділів роботи	19.05.2025-01.06.2025	виконано
6	Попередній захист дипломної роботи	02.06.2025-15.06.2025	виконано
7	Захист дипломної роботи	16.06.2025-22.06.2025	виконано

Студент _____ Андрій ТЕРЕЩУК

Керівник _____ Максим ДРЕВАЛЬ

РЕФЕРАТ

Дана дипломна робота містить 137 с., 21 рис., 10 табл., 2 дод., 24 джерел.

ОСОБИСТА ЕФЕКТИВНІСТЬ, АВТОМАТИЗАЦІЯ, НОТУВАННЯ, КАТЕГОРИЗАЦІЯ, АНАЛІЗ ПРОДУКТИВНОСТІ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОБРОБКА ТЕКСТОВОЇ ІНФОРМАЦІЇ

Об’єктом дослідження є способи можливого підвищення особистої ефективності за допомогою цифрових інструментів, зокрема автоматизації обробки текстових нотаток, їхньої категоризації та аналізу продуктивності користувача.

Предметом дослідження є методи для автоматичної обробки та класифікації особистих нотаток та надання аналітики користувача за його ефективність з метою покращення організації часу та продуктивності.

Метою роботи є розробити програмне забезпечення, яке дозволяє користувачам вести структуровані нотатки з автоматичною категоризацією та аналізом продуктивності.

Актуальність проявляється у зростанні потреби в інструментах для ефективного планування та аналізу особистої продуктивності. Використання алгоритмів обробки тексту для роботи з нотатками дозволяє створити інтелектуальну систему, яка не лише впорядковує інформацію, а й надає цінні аналітичні висновки.

Результатом роботи є створений програмний застосунок на мові C++ та Python, який надає інтерфейс для створення та редагування нотаток, аналізу їх категорії і представлення індивідуальних аналітичних результатів обробки записів.

Подальший розвиток полягає в вдосконалені алгоритмів класифікації, додавання функцій аналізу емоційного забарвлення текстів, створення нових аналітичних модулів, а також покращення інтерфейсу користувача.

ABSTRACT

This diploma thesis contains: 137 p., 21 fig., 10 tabl., 2 app., 24 ref.

PERSONAL PRODUCTIVITY, AUTOMATION, NOTE-TAKING, CATEGORIZATION, PRODUCTIVITY ANALYSIS, SOFTWARE, TEXT PROCESSING

The object of research is to explore possible ways to increase personal productivity using digital tools, particularly the automation of text note processing, categorization, and analysis of user productivity.

The subject of the research is methods for the automatic processing and classification of personal notes, along with providing user analytics on their efficiency, aimed at improving time organization and productivity.

The goal of the work is to develop software that allows users to maintain structured notes with automatic categorization and productivity analysis.

Relevance manifests in the increasing demand for tools that enhance planning and analysis of personal productivity. Utilizing text-processing algorithms for note management facilitates the establishment of an intelligent system that not only organizes information efficiently but also provides valuable analytical insights.

The result of the work is a software application developed in C++ and Python, which provides an interface for creating and editing notes, categorizing them, and presenting individual analytical results of entry processing.

Further development involves improving classification algorithms, adding functions for sentiment analysis of texts, creating new analytical modules, and improving the user interface.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

КПК – кишеньковий персональний комп'ютер (англ. Personal Digital Assistant, PDA).

БД – база даних.

СУБД – система управління базами даних.

API – інтерфейс програмування застосунків (англ. Application Programming Interface).

ACID – атомарність, узгодженість, ізоляція, стійкість (англ. Atomicity, Consistency, Isolation, Durability).

RAII – захоплення ресурсу є ініціалізацією (англ. Resource Acquisition Is Initialization).

MVP – мінімально життєздатний продукт (англ. Minimum Viable Product).

UI – інтерфейс користувача (англ. User Interface).

SOLID – набір з п’яти принципів об’єктно-орієнтованого проєктування (англ. Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), що забезпечують гнучкість, масштабованість і підтримуваність коду.

ВСТУП

У теперішній час багато людей мають безліч різноманітних завдань, які треба тим чи іншим чином організувати, аби досягти потрапляння в дедлайн, підтримати своє ментальне здоров’я, розвинути персональні навички тощо. Для цього більшість людей використовує нотатки – дієвий спосіб організації персональних справ, аби не тримати все в голові і вдало керувати своїм часом, заносячи на матеріальний носій усі важливі для себе помітки.

Спочатку для цього використовувалися паперові носії – вони були першим гарним способом збереження інформації, проте мали безліч недоліків через свою фізичну сутність: людина мусила виписувати кожну нову нотатку вручну без можливості прямого і швидкого перевикористання тексту; було тяжко виправляти помарки в нотатках (виправлення часто могли забруднити нотатку, і першочергова інформація з неї втрачала свій сенс); та й узагалі паперова нотатка у середньо- чи довгостроковій перспективі не була найнадійнішим способом збереження – чорнила з часом вицвітали, а папір був і залишається вразливим до механічних ушкоджень.

З часом з’явилися друкарські машини, які вирішили ряд проблем, пов’язаних з уніфікацією і швидкістю оформлення нотаток. Проте однаково головна інформація зберігалася на папері, з усіма його згаданими недоліками.

У цифрову епоху використання персональних комп’ютерів відкрило широкий спектр можливостей для нотування – від звичайного створення текстового файлу на локальній машині до збереження і синхронізації нотаток у мережевих додатках.

Цей підхід дозволяє оперативно, якісно та структуровано зберігати, змінювати та видаляти інформацію, незалежно від людського почерку та кількості допущених помилок. Хоча цифрові накопичувачі, на яких нотатки зберігаються, самі не вічні і теж піддаються фізичному руйнуванню з часом, цей процес відбувається набагато повільніше. Якщо згадати про існування хмарних

технологій для збереження текстових файлів, то ця проблема взагалі повністю зникає.

Отримавши такий зручний спосіб і перетворивши створення нотатки не на кінцеву ціль, а на частину самого нотування, – почались розробки додатків для розширеного ведення записів. Спочатку це були сервіси для виправлення граматики, маркування тексту, візуальної модифікації нотаток та інше. Проте в останні роки почали з’являтися тенденції більш глибокого нотування з використанням аналітики процесу їх створення та рекомендацій щодо організації своїх записів.

З розвитком штучного інтелекту, а особливо технологій обробки природних мов, для цього напряму відкрились нові можливості, які значно спрощують життя людей і дозволяють краще спланувати свій розпорядок дня, тижня, місяця, року, та, можливо, й ще більших періодів.

У цій роботі буде розглянуто деякі підходи щодо аналізу вмісту нотаток для визначення їхньої категорії на базі вже існуючих, а також впровадження аналітики для оцінки власної ефективності виконання планів у визначені строки, на основі якої користувач зможе отримувати рекомендації щодо покращення створення і затвердження власних дедлайнів.

Створення такого програмного продукту має на меті розробити зручний спосіб оцінки ефективності і отримання рекомендацій щодо планування свого розпорядку за допомогою зручного інтерфейсу і надійного аналітичного модуля.

РОЗДІЛ 1 ОГЛЯД ІСТОРИЧНИХ ТА СУЧАСНИХ МЕТОДІВ І РІШЕНЬ ДЛЯ НОТУВАННЯ ТА ВПОРЯДКУВАННЯ ПЕРСОНАЛЬНИХ ЗАДАЧ

1.1 Актуальність проблеми

Сучасний темп життя вимагає від людей високої організованості, ефективного управління часом, здатності до багатозадачності та можливості структурувати свій життєвий процес для досягнення всіх поставлених задач у визначені терміни та з визначеним рівнем якості виконання. Однак ця перевантаженість життя може призвести до швидкої перевтоми та проблем із ментальним здоров’ям, що в результаті нівелюють будь-яку роботу з організації власного життя людиною та працюють на неї в дисгармонії.

Використання застосунку для нотаток дозволяє спростити ці проблеми, вносячи основну інформацію про свої плани через зручний інтерфейс та маючи до них доступ на постійній основі, не потребуючи розбирати свій почерк чи переписувати вже наявні нотатки з нуля при зміні певних елементів. Для ще більшого підвищення ефективності можна використати автоматичну категоризацію та аналіз продуктивності, що перетворює процес створення нотаток з одностороннього на двосторонній, де користувач на основі своєї активності отримує від програми відповідь і може на базі цього приймати рішення.

Крім того, такий додаток дозволяє людині менше витрачати свій час на вагання щодо встановлення дедлайнів та вибору категорій спірних нотаток персонально для кожного користувача, певним чином навіть автоматизуючи планування свого розкладу, проте залишаючи можливість прийняття остаточного рішення за людиною.

Тому, власне, для вирішення проблеми підвищення особистої ефективності зручним є додаток зі створення нотаток, який поєднував би в собі наступні складові.

1. Швидкий та інтуїтивно зрозумілий інтерфейс для створення, редагування та видалення нотаток із збереженням до бази даних.
2. Функціонал автоматичної класифікації категорії нової нотатки на базі вже наявних категорій різними класифікаторами для представлення оцінок можливої категорії нотатки.
3. Надання статистики щодо виконання нотаток у строки по категоріях, аби розуміти середні часи виконання нотаток і відхилення від них.
4. Оцінка реалістичності виконання завдання в дедлайн, встановлений користувачем для нової нотатки, залежно від його середньої ефективності виконання нотаток вчасно по категорії, а також рекомендації щодо вибору дедлайну для цієї нотатки.

Приклад подібного сервісу і його можливості для аналізу тексту – Evernote – можна побачити на рисунку 1.1.

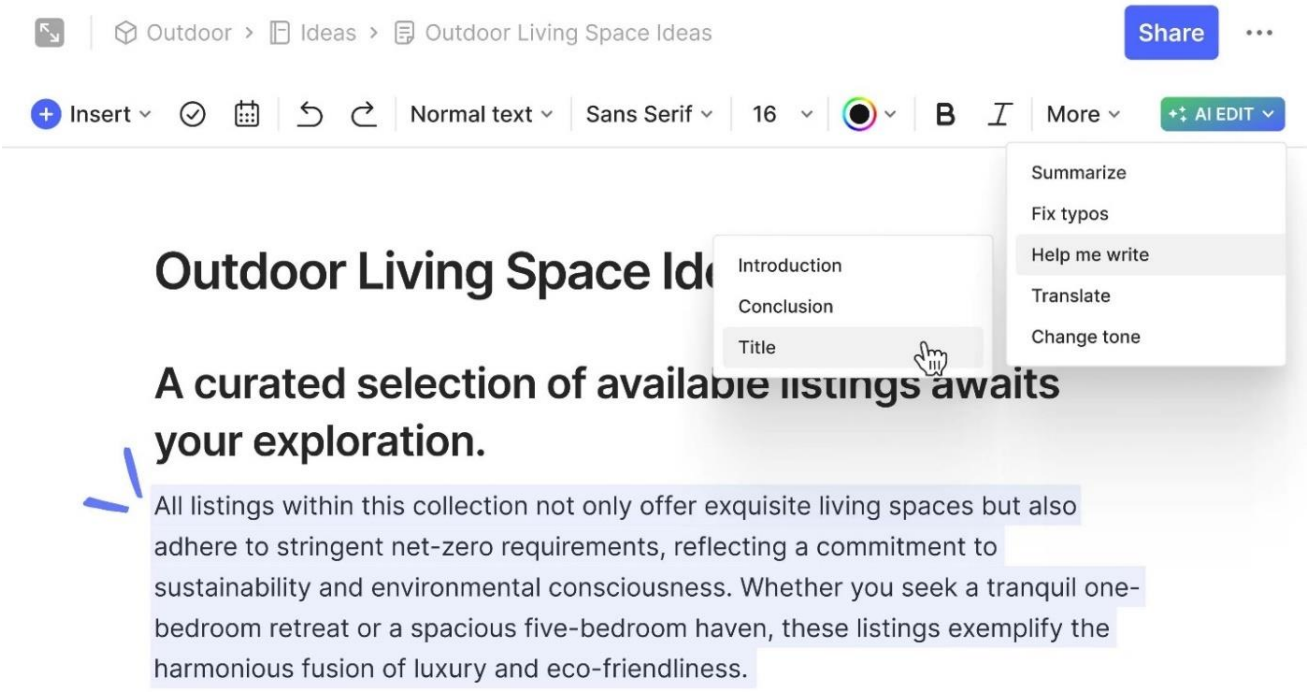


Рисунок 1.1 – Приклад додатку для нотаток з підтримкою аналізу тексту.

Крім очевидної користі для окремих користувачів, застосунок такого типу може бути корисним і в корпоративному середовищі. Продуктові менеджери можуть використовувати такий додаток не лише для власного користування, а й для планування роботи кожного члена проєкту під час спрінтів чи інших організаційних робочих процесів, встановлюючи дедлайни для кожного працівника залежно від його часових показників дотримання дедлайнів.

Додатково до вищезазначених переваг, важливим чинником є можливість накопичення історичних даних користувача, що дозволяє з часом формувати точніші індивідуальні аналітичні моделі. Це забезпечує не лише доцільніші рекомендації щодо планування майбутніх задач, а й сприяє глибшому розумінню особистості користувача додатку: слабких місць у розподілі часу, типових помилок у встановленні дедлайнів, що в результаті відкриває шлях до особистого самовдосконалення на основі об’єктивних даних. Тобто в результаті додаток розвивається разом із людиною, яка його використовує.

Таким чином, розробка застосунку для нотаток з функцією автоматичної категоризації та аналізу продуктивності є актуальною відповіддю на виклики сучасного життя. Вона допомагає вирішити проблему неефективного управління часом та інформацією, забезпечує покращення особистої організованості, сприяє зниженню стресу та підвищенню задоволеності життям. Такий підхід має як особисту, так і комерційну доцільність, що робить його перспективним напрямом для подальшого дослідження і впровадження.

1.2 Історичний розвиток систем нотування та сучасні оптимізаційні рішення

Нотування – один з найзручніших винаходів людства, що має тисячолітню історію. Від часів стародавніх греків і римлян до сучасної цифрової епохи, записування думок і спостережень завжди відігравало важливу роль як у

збереженні знань, так і у підтримці власної ефективності, а також для моделювання своїх планів на майбутні періоди. Тому слід розглянути який розвиток мала ця сфера людської діяльності, а найголовніше – дізнатися, які існують пропозиції щодо перетворення цього процесу на більш комфортний на сучасному цифровому ринку.

1.2.1 Історія становлення сучасного нотування

В джерелі [1] наведено основні етапи історії нотування, його розвитку та цифровізації.

Найдавніші згадки про нотування належать до цивілізацій Стародавньої Греції та Риму. Вони використовували воскові таблички для записів, які були компактними та зручними для багаторазового використання, проте з часом втрачали міцність і були занадто вразливими до механічного руйнування. Філософи, історики, державні діячі та інші інтелектуали того часу фіксували на табличках не лише події, а й особисті міркування, що можна вважати початком історії існування нотування як поняття.

Відкриття паперу в Китаї за часів династії Хань стало справжньою революцією у способах ведення записів. Поряд із папером китайці створили туш і пензлі, що дозволило робити записи швидше та виразніше. Усе це сприяло збереженню знань у централізованих архівах. Також із Китаю традиція нотування поступово поширилася через Шовковий шлях до Ісламського світу та Європи.

У Середньовіччі нотування стало переважно справою монастирів, де ченці вручну переписували релігійні тексти. Вони використовували різноманітні техніки для підвищення точності – прикрашали важливі уривки, писали різнокольоровими чорнилами та робили докладні примітки на полях.

У період Відродження нотування стало більш доступним для широкого загалу. Люди почали використовувати зошити для запису думок і спостережень. Леонардо да Вінчі залишив тисячі сторінок із записами, малюнками й концепціями новаторських ідей. У цей період особисте нотування почало перетворюватися з переважно релігійного інструменту на інструмент самовираження, філософської рефлексії та наукового дослідження. Винахід друкарства також зробив зошити й записники масовим продуктом.

У XVII–XVIII століттях, під час Наукової революції, потреба у фіксації спостережень і експериментів зросла. Учені почали вести наукові журнали, де ретельно описували досліди, робили висновки й систематизували знання. У цей час також виникли академії наук, де нотування стало основою колективної наукової роботи. Запис став обов'язковим етапом експерименту.

XIX століття принесло здешевлення паперу завдяки появі паперових фабрик. Це зробило зошити та реєстри поширеним засобом ведення особистих та фінансових записів. У цей час з'являються масові блокноти, щоденники, бухгалтерські книги. Розвиток поштового зв'язку стимулював ведення листування, що також було формою нотування. У школах нотування почало викладатися як окрема дисципліна, що потребувала додаткового вивчення.

У XX столітті технології почали змінювати спосіб нотування. Винахід друкарської машинки дозволив швидко та акуратно створювати записи. А з появою персональних комп'ютерів і текстових редакторів нотування стало ще більш гнучким – редагувати та впорядковувати записи стало набагато простіше.

Однією з перших спроб вибудовування складних та логічних залежностей під час нотування був метод Zettelkasten (картковий каталог) німецького соціолога Нікласа Лумана. Це дозволяло представляти нотатки у вигляді мережі знань, де ідеї перетиналися й розвивалися, формуючи складні залежності та індекси посилань між собою. Хоча метод був повністю аналоговим за своєю суттю, він став важливою основою для сучасних цифрових систем, таких як Obsidian чи Roam Research.

Справді новітньою була розробка спеціальних персональних цифрових асистентів, які спеціалізувалися на підтримці створення персональних записів користувачів. Цю нішу однією з перших вирішила освоїти компанія Palm, Inc., яка вже в 1996 році представила Pilot 1000 (його вигляд наведено на рисунку 1.2) – КПК, основне призначення якого було у роботі з текстом через надання зручного інтерфейсу для нотування. Вже в ньому був доступний функціонал створення списків планів, розпорядок завдань погодинно на кожен день, синхронізація нотаток та інший доволі прогресивний функціонал на свій час.

Рисунок 1.2 – Pilot 1000.

Відповідно до джерела [2], 30 вересня 2004 року було представлено одну з найперших альтернатив, а саме TiddlyWiki – вікі-двигун з відкритим вихідним кодом, що являє собою вебдодаток/вебсайт, який уміщується на одній вебсторінці, а навігація по ній відбувається за допомогою скриптів. Це рішення надавало можливість користувачам організовувати велику кількість нотаток у вигляді персональної бази знань та за потреби підключати сторонні плагіни у разі необхідності. Проте цей продукт вимагав від користувача відповідної підготовки, оскільки розбір його фреймворку не є тривіальним завданням і потребує певних технічних та програмістських навичок, які є далеко не в кожного, хто хотів би організувати свої нотатки у красивій та зручній формі.

З появою 24 червня 2008 року додатку Evernote цифрове нотування опинилося на новій стадії розвитку. Цей додаток дозволяв зберігати тексти, зображення, аудіо та інші формати у хмарі, синхронізувати їх між пристроями і швидко знаходити за допомогою тегів і пошуку. Його функція розпізнавання зображень була дійсно революційною та неймовірно спрощувала життя користувачів. Проте додаток із часом почав занепадати через стагнацію інтерфейсу та появу конкурентних альтернатив від великого бізнесу, які могли пропонувати більше і швидше на динамічному ринку цифрового нотування.

На теперішній день одним із найпопулярніших рішень є створений 30 березня 2020 року Obsidian – інструмент для нотування, відомий своєю приватністю й широким інструментарієм контролю власного планування. Цей додаток зберігає нотатки у вигляді markdown-файлів локально, на комп'ютері користувача. Потужною перевагою цього додатку є також створення складних мереж зв'язків між нотатками, що дозволяє, використовуючи граф візуалізації, виявляти можливі та неочевидні асоціації між своїми нотатками.

Цифрові технології сьогодні дійсно відкрили всебічні можливості для ведення нотаток. Смартфони, планшети та ноутбуки дозволяють записувати думки в будь-який момент і в будь-якому місці. Інтеграція з календарями, хмарними сервісами та іншими застосунками перетворює нотатки на інтегральну частину щоденного життя людини, без яких її конкурентоспроможність у розвинутих суспільствах буде нижчою за інших людей, які змогли інкорпорувати ці технології у своє персональне та ділове життя.

1.2.2 Сучасні рішення додатків для нотування з аналізом тексту

У наш час у сфері цифрового нотування відбувся справжній прорив завдяки інтеграції штучного інтелекту. Сучасні застосунки вже не просто фіксують інформацію – вони здатні її аналізувати, структурувати та навіть

генерувати новий контент на основі наявних записів, зберігаючи всі напрацювання попередників і покращуючи інтерфейс для взаємодії з доступним функціоналом [3].

Notion AI став одним із найпомітніших інноваторів у цій сфері, адже його головною спеціалізацією є саме нотування з використанням розвинених алгоритмів штучного інтелекту. Програма пропонує цілий набір революційних можливостей: автоматичне створення змістів для довгих документів, аналіз нотаток, виявлення прихованих зв'язків тощо. Система може перетворювати неструктуровані записи на чіткі списки завдань, генерувати ідеї для нотаток на основі вже створених матеріалів або навіть створювати стислий виклад із великих текстів. На сьогодні Notion AI є одним із найрозвинутіших і найпопулярніших рішень для нотування з підтримкою глибокого аналізу контенту. Додатком користуються близько 150 мільйонів осіб щомісяця – як постійні, так і нові користувачі. Річний прибуток компанії від цього продукту становить 300 мільйонів доларів. Все це є індикатором того, що розвиток автоматичного нотування є доволі перспективним напрямком як з точки зору користі, так і з точки зору комерціалізації.

Ще одне запропоноване рішення – застосунок Mem, створений для автоматичної організації нотаток за допомогою машинного навчання. Додаток аналізує контекст записів, визначає ключові теми та поняття, а потім самостійно встановлює зв'язки між різними фрагментами інформації. Особливістю Mem є його здатність певною мірою «передбачати» потреби користувача – наприклад, пропонувати шаблони нотаток на основі вже створених у межах певної категорії.

Roam Research також інтегрував AI-функціональність, хоч і у більш спеціалізованій формі. Його система не лише аналізує текст, а й виявляє логічні зв'язки між концепціями, що є особливо корисним для наукової та дослідницької діяльності. Наприклад, додаток здатен автоматично знаходити повторювані теми у записах або пропонувати дотичні ідеї, які користувач міг не помітити самостійно.

Аналізуючи сучасні інструменти для нотування, стає очевидним, що розвиток AI-можливостей іде у двох ключових напрямках: по-перше, автоматизація рутинних завдань (форматування, категоризація); по-друге, розширення когнітивних можливостей користувача (генерація ідей, виявлення прихованих зв’язків). Навіть популярніші й більш консервативні системи, як Obsidian та Logseq, активно впроваджують штучний інтелект – переважно у формі плагінів (наприклад, AI-плагін для Obsidian) або зовнішніх інтеграцій (наприклад, підключення ChatGPT у Logseq).

Усі ці нововведення дають змогу створювати справжні «мозкові центри», де людина та штучний інтелект працюють у партнерстві. Цікаво, що з розвитком AI-функцій змінюється і сам підхід до ведення нотаток. Якщо раніше користувачі намагалися завчасно структурувати інформацію, продумуючи як зміст, так і архітектуру своїх записів, то тепер системи дозволяють фіксувати думки у довільній формі, а потім – за допомогою аналітичних інструментів – знаходити в них порядок і логічні зв’язки. Це відкриває нові горизонти для творчості, продуктивності та оперативності, перетворюючи процес нотування на інтуїтивну, але водночас й ефективну процедуру.

1.3 Підходи до аналізу тексту

Аналіз тексту базується на різних підходах до представлення його змісту та вимірювання подібності до інших текстів за допомогою метрик відстані. Саме комбінація методів з цих двох груп є основою для будь-яких подальших операцій з текстами: категоризації, побудови рекомендацій, кластеризації тощо.

У цьому розділі буде розглянуто одні з найуживаніших та найпопулярніших, згідно до джерела [4], методів представлення змісту – «торба слів» (Bag of Words), частота терміну-зворотна частота документа (TF-IDF) та

латентно-семантичний аналіз (LSA), а також основні метрики для вимірювання подібності між текстами: евклідова, косинусна та мангеттенська відстані.

1.3.1 Торба слів (Bag of Words)

Один з найпростіших і найперших запропонованих методів для векторизації тексту, який був запропонований ще в 1950-60-х роках. Текст перетворюється на набір окремих слів без урахування їх порядку і граматики. Цей метод доволі часто використовують для класифікації документів, де кожне слово має вагу, яке відповідає кількості його появ у тексті. Зокрема цей метод використовували перші пошукові системи (з відомих – рушій пошуку Yahoo!). Цей підхід має ряд переваг.

1. Метод простий у реалізації.
2. Швидкий для невеликих обсягів текстової інформації.
3. Добре працює для базової класифікації.

Цей підхід проте має й ряд недоліків.

1. Повністю ігнорує порядок слів і їх контекст в текстовій інформації.
2. Повністю ігнорує важливість слова в текстовій інформації.
3. При збільшенні текстової інформації вектор стає більш розрідженим.

1.3.2 TF-IDF (Term Frequency - Inverse Document Frequency)

У 1970-х роках дослідження в галузі інформаційного пошуку активізувалися, зокрема в напрямку покращення підходу BoW. Однією з основних проблем було те, що часто використовувані службові слова («і», «на», «у» тощо) не несли змістовного навантаження, але впливали на результати.

Карен Спарк Джонс у своїй статті 1972 року вперше чітко сформулювала концепцію зважування слів у залежності від їхньої частотності.

TF-IDF – це метод зважування слів, який складається з методів TF і IDF.

1. TF (частотність слова в документі) – наскільки часто слово зустрічається в тексті.
2. IDF (зворотна частота документа) – наскільки слово рідкісне серед усіх документів.

Цей підхід дозволяє підкреслити важливі для конкретного тексту слова, ігноруючи надто загальні терміни. Цей підхід має ряд переваг.

1. Покращує BoW, надаючи ваги словам.
2. Добре працює для пошуку та ранжування документів.
3. Порівняно простий в інтерпретації і в імплементації.

Цей підхід проте має й ряд недоліків.

1. Все ще не враховує семантику текстової інформації.
2. Так само при великих обсягах інформації стає більш розрідженим, хоч і не так швидко, як в методі мішку слів.

1.3.3 Латентно-семантичний аналіз

Latent semantic analysis – це один із перших методів тематичного моделювання текстів, запропонований у 1990-х роках. Метод швидко набув популярності в інформаційному пошуку та обробці природної мови завдяки здатності виявляти латентні семантичні зв’язки між словами та документами.

Латентно-семантичний аналіз базується на побудові матриці частот появи термінів у документах (наприклад, TF-IDF), після чого застосовується сингулярне розкладання для зниження розмірності цієї матриці. Таким чином виділяється кілька латентних (прихованих) тем, які узагальнюють семантику текстів.

Після трансформації як документи, так і слова проєктуються у спільний простір, що дозволяє аналізувати подібність за змістом, а не лише за лексичною подібністю. Цей підхід має ряд переваг.

1. Враховує латентні семантичні зв’язки між словами та документами.
2. Покращує пошук за змістом, навіть якщо точні слова не збігаються.
3. Знижує розмірність простору, позбавляючись шуму в даних.

Цей підхід проте має й ряд недоліків.

1. Лінійна модель – не враховує складні структури мови.
2. Вразливий до шуму та рідкісних термінів без належної обробки.

1.3.4 Евклідова текстова відстань

Цей підхід базується на однойменному математичному методі для визначення відстані. В цих задачах текст перетворюють на числовий вектор (наприклад, за частотою слів), а потім рахують відстань між цими векторами. Цей підхід має ряд переваг.

1. Простота імплементації.
2. Добре працює, коли тексти мають однакову довжину.

Цей підхід проте має й ряд недоліків.

1. Чутливість до великих чисел (наприклад, якщо одне слово зустрічається доволі часто, воно може перекрити інші).
2. Не враховує семантику (два тексти можуть бути близькі за частотами слів, але з різним змістом).

1.3.5 Косинусна текстова відстань

Метод прийшов із лінійної алгебри та активно використовується в машинному навчанні. Якщо згадати, що ми представляємо текст як вектор, то нам не так сильно важливий розмір вектора (кількість слів), а більше цікавить власне його «напрямок». Цей підхід має ряд переваг.

1. Незалежний від довжини тексту.
2. Добре ловить семантичні зв’язки.

Цей підхід проте має й ряд недоліків.

1. Не враховує рідкісні, але важливі слова, особливо якщо вони є в обох текстах, проте в малих кількостях.
2. Потребує нормалізованих векторів.

1.3.6 Мангеттенська текстова відстань

Названий через схожість з плануванням вулиць Мангеттена, де потрібно йти тільки горизонтально або вертикально. У текстовій аналітиці рахується не пряма відстань, а сума всіх «кроків» по кожній компоненті вектора. Цей підхід має ряд переваг.

1. Менш чутливий до викидів (якщо одне слово різко відрізняється, то це не так сильно впливає).
2. Краще працює з розрідженими даними.

Цей підхід проте має й ряд недоліків.

1. Менш інтуїтивна для візуалізації.
2. Не завжди добре працює для семантичної схожості.

1.4 Способи створення аналітики і передбачення дати на основі минулих періодів

Аналітика продуктивності та прогнозування подій на основі історичних даних є ключовими елементами для створення інтелектуального додатку, спрямованого на покращення особистої ефективності. У джерелі [5] наведені можливості аналізу часових показників для створення передбачень, а тому використання минулих періодів дозволяє виявити закономірності у поведінці користувача, автоматизувати планування та надавати рекомендації щодо оптимальних термінів виконання задач.

1.4.1 Базові статичні показники

Середнє значення, медіана та стандартне відхилення можуть бути використані для опису типової продуктивності користувача. Вони дають початкову картину того, скільки часу в середньому займає виконання завдань, який час є найбільш поширеним та наскільки цей час коливається.

Середнє значення добре підходить для даних без екстремальних значень, але воно чутливе до викидів.

Медіана ігнорує викиди та відображає центральне значення вибірки, що робить її надійнішою для аналізу асиметричних розподілів.

Стандартне відхилення показує, наскільки значення в середньому відхиляються від середнього, що дає змогу оцінити стабільність поведінки.

Ці показники є фундаментом для подальшого моделювання.

1.4.2 Лінійна регресія

Лінійна регресія використовується для моделювання зв’язку між кількістю слів у нотатці та часом її виконання. Такий аналіз дозволяє встановити, чи більша довжина тексту призводить до збільшення ймовірності порушення дедлайну.

Результати регресії можна інтерпретувати зрозуміло для користувача, наприклад коефіцієнт детермінації показує, наскільки добре модель пояснює дані, а p-value вказує на статистичну значимість зв’язку.

Цей метод працює лише для лінійних залежностей. Якщо реальний зв’язок між кількістю слів і часом виконання нелінійний, то модель даватиме неточні прогнози.

1.4.3 Логістична регресія

Логістична регресія – це статистичний метод, який допомагає передбачити ймовірність настання події (наприклад, виконання завдання вчасно) на основі однієї або кількох незалежних змінних. У контексті аналітики дедлайнів вона використовується для оцінки того, як запас часу (різниця між дедлайном і датою створення завдання) впливає на ймовірність успішного завершення.

Проте можлива лінійність припущень – модель приймає, що зв’язок між незалежними змінними та логарифмом шансів є лінійним. Якщо запас часу має нелінійний характер, модель це не врахує. А також можливе ігнорування контексту – якщо запас часу по-різному впливає на різні категорії завдань, звичайна логістична регресія не виявить таких нюансів без явного включення взаємодій між змінними.

1.4.4 Кореляційний аналіз

Кореляційний аналіз допомагає знайти зв’язки між параметрами завдань. Для задач типу аналізу нотування можна використовувати два типи кореляції.

1. Коефіцієнт Пірсона – для зв’язку між двома числовими змінними (наприклад, кількість слів і час виконання). Отримаємо значення в діапазоні $[-1,1]$, які мають відповідну інтерпретацію:
   1) 1 – ідеальний позитивний зв’язок;
   2) -1 – ідеальний негативний зв’язок;
   3) 0 – відсутність зв’язку.
2. Точковий бісеріальний коефіцієнт – для зв’язку між числовою змінною (запас часу) та бінарною (факт виконання). Залежно від отриманого коефіцієнту (більше чи менше 0), ми маємо відповідну оцінку того, як більше часу впливає на швидкість виконання нотатки (коефіцієнт переводиться у відсоткову ймовірність виконання нотатки швидше/повільніше залежно від знаку).

Важливо відмітити, що кореляція не завжди означає причинність. Навіть сильний зв’язок не доводить, що одна змінна викликає зміни в іншій.

1.4.5 Дисперсійний аналіз

ANOVA (дисперсійний аналіз) використовується для порівняння середніх значень трьох або більше груп. У контексті нотування це може бути порівняння часу виконання завдань, створених у різний час доби (ранок, день, вечір, ніч). Формуються відповідні гіпотези для нашого варіанта.

1. Нульова гіпотеза (H0) – середні часи виконання для всіх груп однакові.
2. Альтернативна гіпотеза (H1) – принаймні одна група відрізняється.

Відповідно до них формуються результати обчислень.

1. F-статистика – показує, наскільки міжгрупова варіація перевищує внутрішньогрупову.
2. p-value – якщо воно нижче певного значення, то нульова гіпотеза відхиляється – існують суттєві відмінності.

1.4.6 Оцінка запасу часу

Z-значення використовуються для розрахунку запасу часу, який додається до прогнозованого часу виконання, щоб врахувати природну варіативність даних. Це дозволяє зменшити ризик прострочення.

Використовуючи середнє значення та стандартне відхилення часу виконання нотаток з тієї ж категорії, можна розрахувати рекомендований дедлайн із врахуванням певного рівня довіри.

1.5 Висновки до першого розділу

Сучасні темпи життя підвищують потребу в ефективних інструментах для організації інформації й управління особистою продуктивністю. Застосунки для нотування з автоматичним аналізом і категоризацією допомагають користувачам упорядковувати записи, швидше знаходити потрібну інформацію й краще розуміти власну активність. Сьогодні інтелектуальні додатки активно використовують методи аналізу тексту, машинного навчання й обробки природної мови для покращення користувацького досвіду. На ринку вже існують рішення, що підтверджують актуальність і затребуваність подібних застосунків, а тому розвиток цієї сфери має високий потенціал.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗОВАНОЇ КАТЕГОРИЗАЦІЇ НОТАТОК Й АНАЛІЗУ ПЕРСОНАЛЬНОЇ ПРОДУКТИВНОСТІ, ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.

2.1 Вибір метрик для передбачення категорії нотаток

Попри наявність великої кількості методів для визначення категорії тексту, деякі з них демонструють стабільні результати частіше за інші, що зумовило їх ширше використання на практиці. Саме ці методи було обрано для реалізації в програмному рішенні з метою їх комбінації та отримання зваженого прогнозу. Вибір обґрунтовано їхньою популярністю, перевіреною ефективністю та стабільністю під час застосування в задачах. Наведені підходи базуються на джерелі [6], де категоризація нотаток, використовуючи відповідні алгоритми, є лише частиною можливого прикладного застосування наведених алгоритмів.

2.1.1 Bag of words з євклідовою, чебишевською і мангеттенською відстаннями

Метод класифікації текстів на основі розподілів частот слів і метрик відстані є одним із найпростіших і найбільш інтуїтивно зрозумілих підходів до автоматичної категоризації текстових даних. Цей метод ґрунтується на концепції, що тексти, які належать до однієї категорії, мають схожі профілі вживання слів.

У цьому підході кожен текст представляється як нормалізований вектор частот слів (модель мішку слів), а категорії – як середньозважені розподіли слів на основі тренувальних даних. Для нового тексту обчислюється відстань між

його розподілом слів і розподілами кожної категорії. Категорія, чий профіль вживання слів найбільш подібний до тексту (має найменшу відстань), обирається як найближча за сенсом до нової нотатки.

Нехай у нас є набір документів з множиною унікальних слів $W = \{w_1, w_2, \dots, w_n\}$ кількістю n. Для документа d створюємо вектор частот i-го слова в документі d ($f_{d,i}$) в вигляді $\overrightarrow{v_d} = \{f_{d,1}, f_{d,2}, \dots, f_{d,n}\}$.

Потім проводимо нормалізацію, щоб отримати відносні частоти за формулою 2.1.

$$f_{d,i} = \frac{c_{d,i}}{\sum_{j=1}^{n} c_{d,j}}, \tag{2.1}$$

де $c_{d,i}$ – кількість входжень слова w_i в документ d.

Ця нормалізація дозволяє порівнювати документи різної довжини.

Для кожної категорії k з множини категорій K маємо набір документів $D_k = \{{d_1}^k, {d_2}^k, \dots, {d_{m_k}}^k\}$, де m_k – кількість документів у категорії k.

Агрегований частотний вектор для категорії k представляємо як $\overrightarrow{v_k} = \{f_{k,1}, f_{k,2}, \dots, f_{k,n}\}$, де $f_{k,i}$ знаходимо за формулою 2.2.

$$f_{k,i} = \frac{\sum_{j=1}^{m_k} {c_{d_j}}^{k,i}}{\sum_{j=1}^{m_k} \sum_{l=1}^{n} {c_{d_j}}^{k,l}}. \tag{2.2}$$

Тобто спочатку підсумовуємо всі входження слова w_i у всіх документах категорії k, а потім ділимо на загальну кількість слів у всіх документах категорії.

Після цього знаходимо відстані між частотами по категоріям і новою нотаткою за однією з можливих формул відстаней.

Манхеттенська відстань (L_1-норма), формула 2.3.

$$d_M(\vec{a}, \vec{b}) = \sum_{i=1}^{n} |a_i - b_i|. \tag{2.3}$$

Евклідова відстань (L_2-норма), формула 2.4.

$$d_E(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}. \tag{2.4}$$

Відстань Чебишева (L_∞-норма), формула 2.5.

$$d_E(\vec{a}, \vec{b}) = \max_{i=1,2,\dots,n} |a_i - b_i|. \tag{2.5}$$

Тобто загальний підхід для визначення категорії нової нотатки будується за чітко визначеним алгоритмом.

1. Створюємо його частотний вектор.
2. Для кожної категорії обчислюємо відстань з частотним вектором цієї нотатки.
3. Перетворюємо відстані на подібності через нормалізацію.
4. Обираємо категорію з найбільшою подібністю.

Підсумовуючи – цей підхід добре застосовувати для невеликих і однорідних груп текстів із чіткими категоріями. Однак для складніших задач, де важливий контекст, синонімія чи варіативність мови, цей підхід поступається сучаснішим методам (наївний баєс, TF-IDF, LSA).

2.1.2 TF-IDF з косинусовою відстанню

Цей метод належить до класичних алгоритмів інформаційного пошуку та обробки природної мови, і широко використовується у різноманітних задачах текстової класифікації, включаючи фільтрацію спаму, категоризацію новин та аналіз настроїв [7].

На відміну від простих частотних методів, TF-IDF забезпечує більш тонке врахування важливості термінів у документі, зважуючи частоту слова в документі на обернену частоту його появи в групі документів, таким чином знижуючи вагу загальновживаних слів і підвищуючи значущість термінів, характерних для конкретної теми. А використання косинусової подібності є найліпшим способом визначення подібностей між текстами для TF-IDF, як, наприклад, у джерелі [8].

Комбінуються дві метрики.

1. Term Frequency (TF) – як часто слово зустрічається в конкретній нотатці.
2. Inverse Document Frequency (IDF) – наскільки рідкісним є слово в усіх нотатках.

Для кожного документа d і кожного слова t в ньому, TF-IDF обчислюється як добуток двох компонент, наведених в формулі 2.6.

$$TF - IDF(t, d) = TF(t, d) * IDF(t). \quad (2.6)$$

Тут TF вимірює, наскільки часто терм t зустрічається в документі d, нормалізований за довжиною документа, що наведено в формулі 2.7.

$$TF(t, d) = \frac{f(t,d)}{|d|}, \quad (2.7)$$

де $\mathrm{f(t, d)} -$ асболютна частота терміну t в документі d;

$\mathrm{|d|} -$ кількість слів у документі d.

IDF вимірює, наскільки унікальним або рідкісним є терм t у всій множині документів D, що наведено в формулі 2.8.

$$IDF(t) = \log\left(\frac{|D|}{|\{d \in D: t \in d\}|}\right), \quad (2.8)$$

де $|D|$ – загальна кількість документів у множині;

$|\{d \in D: t \in d\}|$ – кількість документів, що містять термін t.

В контексті категоризації нотаток виконуються операції.

1. Для категорії c з набором документів D_c формуємо вектор категорій за TF-IDF значеннями всіх слів у всіх документах відповідної категорії $\overrightarrow{v_c} = \{TF - IDF(t_1, D_c), TF - IDF(t_2, D_c), \dots, TF - IDF(t_n, D_c)\}$.
2. Для нового документа d_{new} з вектором TF-IDF $\vec{v}_{d_{new}}$, обчислюється косинусна подібність (формула 2.9) за кожною категорією.

$$similarity(d_{new}, c) = \frac{\vec{v}_{d_{new}} \cdot \overrightarrow{v_c}}{||\overrightarrow{v_{d_{new}}}|| * ||\overrightarrow{v_c}||}, \quad (2.9)$$

де $\vec{v}_{d_{new}} \cdot \overrightarrow{v_c}$ – скалярний добуток векторів;

$||\overrightarrow{v_{d_{new}}}|| * ||\overrightarrow{v_c}||$ – евклідові норми (довжини) векторів.

Косинусна подібність приймає значення в діапазоні $[-1, 1]$, де:

1) 1 означає ідентичне напрямок векторів (максимальна подібність);
2) 0 означає ортогональність (відсутність подібності);
3) -1 означає протилежний напрямок (максимальна відмінність).

Для невід'ємних векторів, якими є вектори TF-IDF, діапазон обмежується $[0, 1]$.

Прогнозована категорія визначається як та, що має максимальну косинусну подібність.

TF-IDF з косинусною подібністю залишається одним із найбільш надійних і широко використовуваних підходів до класифікації текстів. Цей метод часто служить міцним базовим алгоритмом, з яким порівнюють продуктивність нових, складніших моделей. Його ефективність, масштабованість та інтерпретованість роблять його незамінним інструментом у багатьох сферах обробки природної мови.

2.1.3 Наївний байєсівський класифікатор

Наївний байєсівський класифікатор представляє собою ймовірнісний метод для категоризації текстів, що базується на фундаментальній теоремі Байєса. Незважаючи на свою математичну простоту, цей алгоритм демонструє ефективність у багатьох практичних задачах. Його основна ідея полягає у обчисленні ймовірностей належності документа до різних категорій з використанням статистичних даних про частоту появи слів у наданих документах.

Спочатку визначаємо поширеність кожної категорії у нашій вибірці за формулою 2.10.

$$P(C = c_j) = \frac{N_{c_j}}{N}, \tag{2.10}$$

де N_{c_j} – кількість документів категорії c_j у вибірці N;

N – загальна кількість документів.

Далі для кожного слова w_i і категорії c_j обчислюється умовна ймовірність (формула 2.11).

$$P(w_i|C = c_j) = \frac{count(w_i, c_j)}{count(c_j)}, \tag{2.11}$$

де $\mathrm{count}(\mathrm{w_i}, \mathrm{c_j})$ – скільки разів слово $\mathrm{w_i}$ трапляється в документі категорії $\mathrm{c_j}$;

$\mathrm{count}(\mathrm{c_j})$ – загальна кількість слів у документах категорії $\mathrm{c_j}$.

Для вирішення проблеми слів, які не зустрічались у навчальних даних, застосовується згладжування Лапласа (формула 2.12).

$$P(w_i|C = c_j) = \frac{count(w_i, c_j) + \alpha}{count(c_j) + \alpha|V|}, \tag{2.12}$$

де α – параметр згладжування;

$|\mathrm{V}|$ – розмір словника (кількість унікальних слів).

Для нового документа d_{new} ймовірність його належності до категорії $\mathrm{c_j}$ обчислюється за формулою 2.13.

$$P(C = c_j|d_{new}) \sim P(C = c_j) \prod_{i=1}^{n} P(w_i|C = c_j). \tag{2.13}$$

Для уникнення проблеми зникаючих значень при множенні малих ймовірностей, обчислення проводяться в логарифмічній шкалі (формула 2.14).

$$\log\left(P(C = c_j|d_{new})\right) \sim \log\left(P(C = c_j)\right) + \sum_{i=1}^{n} \log\left(P(w_i|C = c_j)\right). \tag{2.14}$$

Для класифікації вибирається категорія з найбільшою ймовірністю.

Додатково можна нормалізувати отримані ймовірності, попередньо позбувшись логарифму через експоненціювання.

2.1.4 Множинна схожість за індексом Жаккара

Цей метод належить до родини алгоритмів, що використовують порівняння наборів слів (без урахування їхньої частоти) для визначення категорії документа. На відміну від частотних методів, підхід на основі індексу Жаккара фокусується виключно на присутності чи відсутності слів у документі, а не на кількості їх входжень.

Для документа d створюється множина унікальних слів $S_d = \{w_1, w_2, \dots, w_n\}$, де w_i – унікальне слово, що зустрічається в документі після видалення несуттєвих слів.

Для категорії c, яка містить набір документів $D_c = \{{d_1}^c, {d_2}^c, \dots, {d_m}^c\}$, формується множина всіх унікальних слів категорії $S_c = S_{{d_1}^c} \cup S_{{d_2}^c} \cup \dots \cup S_{{d_n}^c}$.

Для нового документа d_{new} з відповідною множиною слів $S_{d_{new}}$ обчислюється коефіцієнт Жаккара за кожною категорією (формула 2.14).

$$J(S_{d_{new}}, S_c) = \frac{|S_{d_{new}} \cap S_c|}{|S_{d_{new}} \cup S_c|}, \tag{2.14}$$

де $|S_{d_{new}} \cap S_c|$ – кількість елементів, що належать одночасно до нової та архівної нотатки;

$|S_{d_{new}} \cup S_c|$ – кількість елементів, що належать хоча б до однієї з нотаток.

Математично, коефіцієнт Жаккара набуває значень від 0 до 1.

1. $J(S_{d_{new}}, S_c) = 0$, якщо множини не мають спільних елементів.
2. $J(S_{d_{new}}, S_c) = 1$, якщо множини ідентичні.

Прогнозована категорія визначається як та, що має максимальну подібність за Жаккаром.

Незважаючи на свою простоту, метод Жаккара часто використовується як базовий підхід у задачах текстової класифікації та може служити основою для більш складних алгоритмів. У поєднанні з якісною попередньою обробкою тексту він може показувати досить високу точність для певних типів документів.

2.1.5 Латентно-семантичний аналіз з косинусовою відстанню

Методи Bag of Words та TF-IDF дозволяють формувати векторні подання документів, однак ці подання залишаються високорозмірними та поверхневими – вони не враховують глибинних семантичних зв'язків між словами. Щоб подолати це обмеження, було розроблено метод Latent Semantic Analysis (LSA), який зберігає структуру TF-IDF, але додає латентне (приховане) тематичне узагальнення через пониження розмірності [9].

Маємо набір документів $D = \{d_1, d_2, \dots, d_N\}$, кожен представлений у вигляді набору термінів. Після цього відбувається побудова матриці з використанням методу TF-IDF за формулою 2.15.

$$X_{ij} = TF(t_j, d_i) * IDF(t_j). \tag{2.15}$$

В результаті маємо $X \in R^{N \times V}$, матрицю для N документів та V термінів.

Після цього відбувається сингулярний розклад матриці X за формулою 2.16.

$$X \approx U_k \Sigma_k {V_k}^T, \tag{2.16}$$

де $U_k \in R^{N \times k}$ – матриця документів у зниженому просторі (ліва сингулярна матриця);

$\Sigma_k \in R^{k \times k}$ – діагональна матриця сингулярних значень;

${V_k}^T \in R^{k \times V}$ – матриця термінів у зниженому просторі (права сингулярна матриця).

Це дозволяє проектувати як документи, так і слова в спільний k-вимірний латентний семантичний простір.

Маємо для кожного документа його координати $U_k[i]$. Якщо і-ий документ належить до категорії c, то вектор категорії c визначається за формулою 2.17 як середнє значення всіх векторів документів цієї категорії:

$$\vec{C}_c = \frac{\sum_{d_i \in D_c} U_k[i]}{|D_c|}, \tag{2.17}$$

де D_c – множина документів категорії c.

Для нового документа d проводимо обчислення його TF- IDF вектору ($\vec{d}$) і проектуємо у зменшений простір (ділимо на Σ_k для нормалізації на сингулярні значення) за формулою 2.18.

$$\vec{v} = \frac{\vec{d} \cdot {V_k}^T}{\Sigma_k}. \quad (2.18)$$

Обирається категорія c, що має найбільшу косинусну подібність до вектора нового документа.

LSA дозволяє не просто рахувати частоти слів, а виявляти теми та приховані зв’язки у текстах, що робить його значно глибшим за класичні методи «мішка слів» та TF-IDF. Він добре працює для класифікації, пошуку та кластеризації текстів, знижуючи шум і підвищуючи якість семантичного узагальнення.

2.1.6 Агреговані методи

Для отримання декількох варіантів для обрання використовується об'єднання результатів декількох моделей за принципом голосування або обчислення середнього передбачення. Для голосування підсумкова категорія обирається за формулою 2.19.

$$C = arg \max_{c} \sum_{i=1}^{k} \mathbb{1}(C_i = c), \quad (2.19)$$

де C_i – категорія, передбачена і-ю моделлю;

k – кількість класифікаторів;

$\mathbb{1}$ – індикаторна функція.

Такий підхід дозволяє компенсувати недоліки окремих моделей і забезпечити більш стабільні результати на різноманітних типах нотаток.

Таким чином, вибрані метрики і методи дозволяють ефективно обробляти тексти нотаток, забезпечуючи варіативність та надаючи користувачу можливість приймати рішення, отримуючи рекомендації від відповідних методів.

2.2 Процес аналізу часу виконання нотаток

Цей підхід передбачає збір даних про тривалість виконання задач, їх категорії та контекст, що дозволяє виявити тенденції. Використання статистичних моделей та візуалізацій (графіки, гістограми) перетворює сирі метрики на зрозумілі звіти.

2.2.1 Зведена статистика активності користувача за категоріями

Цей функціонал має на меті надати користувачу базову інформацію щодо його нотування. Для цього представимо наші нотатки у вигляді множини $N = \{n_1, n_2, \dots, n_k\}$, де кожна нотатка n_i має 4 головні атрибути.

1. $c_i \in C$ – категорія нотатки.
2. $s_i \in \{\text{«todo»}, \text{«done»}\}$ – статус нотатки.
3. $t_i \in R_{\geq 0} \cup \{\emptyset\}$ – час завершення (якщо статус done).
4. $a_i \in R \cup \{\emptyset\}$ – показник дотримання дедлайну (може бути від'ємним або позитивним).

Для подальшого аналізу виконується групування нотаток за категоріями $G_c = \{n_i \in N : c_i = c\},\ \forall c \in C$.

Для кожної категорії знаходимо відповідні статистичні показники.

Кількість нотаток (формула 2.20).

$$N_c = |G_c|. \tag{2.20}$$

Розподіл статусів (формула 2.21).

$$freq_c(s) = |\{n_i \in N : s_i = s\}|. \quad (2.21)$$

Рівень завершення (формула 2.22).

$$completion\_rate_c(s) = \frac{freq_c(done)}{freq_c(done)+freq_c(todo)}. \quad (2.22)$$

Формуємо множину показників для часу завершення $T_c = \{t_i : n_i \in G_c, s_i = done, t_i \neq \emptyset\}$. І статичні показники для нього.

Середній час (формула 2.23).

$$\mu_t = \frac{1}{|T_c|}\sum_{t \in T_c} t. \quad (2.23)$$

Медіана (формула 2.24).

$$median_t = med(T_c). \quad (2.24)$$

Стандартне відхилення (формула 2.25).

$$\sigma_t = \sqrt{\frac{1}{|T_c|}\sum_{t \in T_c}(t - \mu_t)^2}. \quad (2.25)$$

Так само описуємо множину для дотримання дедлайнів $A_c = \{a_i : n_i \in G_c, s_i = done, a_i \neq \emptyset\}$ і для неї формуємо аналогічні формули 2.23, 2.24 та 2.25.

Додатково обчислимо кількість вчасно виконаних нотаток (формула 2.26).

$$on\_time\_rate_c = \frac{|\{a \in A_c : a \geq 0\}|}{|A_c|}. \quad (2.26)$$

2.2.2 Аналіз швидкості виконання нотаток

Цей функціонал має на меті надати користувачу звіт з аналізу швидкості виконання його нотаток залежності від кількості слів і часу їх виконання. Для цього формуємо множину завершених нотаток $D = \{n_1, n_2, \dots, n_m\}$, де кожна нотатка n_i має 4 головні атрибути.

1. $w_i \in N$ – кількість слів у тексті.
2. $c_i \in C$ – категорія нотатки.
3. $t_i \in R_{\geq 0} \cup \{\emptyset\}$ – час завершення.

4. $h_i \in H = \{morning, afternoon, evening, night\}$ – частина доби створення нотатки.

Для знаходження кореляції між кількістю слів і часом виконання використовуємо коефіцієнт Пірсона (формула 2.27).

$$r = \frac{\sum(w_i - \bar{w})(t_i - \bar{t})}{\sqrt{\sum(w_i - \bar{w})^2}\sqrt{\sum(t_i - \bar{t})^2}}, \quad (2.27)$$

де $\bar{w}$ та $\bar{t}$ – середні значення кількості слів і часів завершення.

Далі для кількості слів і часу виконання знаходимо лінійну регресію за формулою 2.28.

$$\widehat{t_i} = \beta_0 + \beta_1 w_i, \quad (2.28)$$

де β_0 – зміщення;

β_1 – коефіцієнт нахилу.

Результатом обчислення якої будуть 3 параметри.

1. R^2 – коефіцієнт детермінації.
2. std_err – стандартна похибка
3. $p - value$ – значення для тесту значущості регресії.

Далі для кожної категорії і часу доби обчислюємо середні швидкості виконання (формули 2.29 і 2.30 відповідно):

$$\mu_c = E\left[\frac{t_i}{w_i}\right], \ \forall i: c_i = c, \quad (2.29)$$

$$\mu_h = E\left[\frac{t_i}{w_i}\right], \ \forall i: h_i = h. \quad (2.30)$$

Після цього для часу створення нотатки використовуємо ANOVA (analysis of variance), формуємо нульову гіпотезу H_0, яка каже, що усі часові групи нотаток однакові за своєю ефективністю, тобто $\mu_i = \mu_j$. За альтернативну гіпотезу H_1 беремо, що існують хоча б якісь дві групи нотаток, що $\mu_i \neq \mu_j$.

В результати обчислюємо F-статистику, p-значення і рішення про значущість на основі заданого порогу.

Також проводимо виявлення викидів, тобто відхилення від часу які знаходяться за нормальними межами використовуючи формулу 2.31.

$$d_i = t_i - \bar{t}. \quad (2.31)$$

Потім проводимо перевірку на те, чи це дійсно викид за формулою 2.32.

$$d_i > Q3 + 1.5 * IQR, \tag{2.32}$$

де $Q1 =$ 25-та перцентиль;

$Q3 =$ 75-та перцентиль;

$IQR = Q3 - Q1$.

2.2.3 Аналіз дотримання дедлайнів

В цьому функціоналі передбачається надання детальної статистики по дотриманню дедлайнів користувачем.

Для цього формуємо чотири множини.

1. N – усі нотатки.
2. $N_D \subset N$ – нотатки, які мають дедлайн.
3. $N_C \subset N_D$ – нотатки з дедлайном, які завершені.
4. $N_{OT} \subset N_C$ – завершені нотатки, де дотримано дедлайн.

Відповідно кожна нотатка має 6 головних параметрів.

1. d_n – дедлайн.
2. c_n – дата завершення.
3. s_n – дата створення.
4. $a_n = (d_n - c_n)$ – час між встановленим дедлайном і датою завершення нотатки.
5. $b_n = (d_n - s_n)$ – час між встановленим дедлайном і датою створення нотатки.
6. $y_n = \mathbb{1}(n \in N_C)$.

Далі обраховуємо загальні метрики.

Рівень завершення (формула 2.33).

$$R_c = \frac{|N_C|}{|N_D|}. \tag{2.33}$$

Рівень вчасності (формула 2.34).

$$R_{ot} = \frac{|N_{OT}|}{|N_D|}. \tag{2.34}$$

Середнє відхилення від дедлайну (формула 2.35).

$$\bar{a} = \frac{\sum_{n \in N_C} a_n}{|N_C|}. \tag{2.35}$$

Додатково формуємо для кожної категорії множину $A_k = \{a_n | n \in N_C \wedge category(n) = \kappa\}$.

І для неї аналогічно знаходимо середнє, медіану та стандартне відхилення за формулами 2.33 – 2.35.

Проводимо експеримент на те, скільки буфер часу впливає на завершення нотатки, використовуючи точково-бісеріальну кореляцію (формула 2.36).

$$r_{pb} = \frac{\overline{b_1} - \overline{b_0}}{s_b} \sqrt{\frac{n_1 n_0}{n(n-1)}}, \tag{2.36}$$

де $\overline{b_1}$ – середній буфер у нотатках, що були завершені;

$\overline{b_0}$ – середній буфер у нотатках, що не були завершені;

s_b – стандартне відхилення буфера часу серед усіх нотаток;

n_1, n_0 – кількість завершених і незавершених нотаток відповідно;

$n = n_1 + n_0$ – загальна кількість нотаток з дедлайнами.

Остаточно формуємо ймовірність завершення як функцію буфера b_n в вигляді логістичної регресії (формула 2.37).

$$P(y_n = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 b_n)}}, \tag{2.37}$$

де β_0 – зміщення;

β_1 – коефіцієнт нахилу.

2.2.4 Рекомендації щодо дедлайну

Важливим елементом цього модулю є рекомендація нотатки за загальною статистикою. Для цього проводиться зважена оцінка декількох способів передбачення.

Спочатку вводимо 4 окремі параметри.

1. n – кількість завершених нотаток у вибраній категорії.
2. w – кількість слів у новій нотатці.
3. w_i – кількість слів у нотатці і.
4. t_i – час (у годинах), витрачений на завершення нотатки і.

Ці значення нам знадобляться для формування відповідних результатів програмного передбачення.

По-перше, знаходимо середній час завершення за формулою 2.37.

$$\bar{t} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} t_i. \tag{2.37}$$

Медіану часів завершення за формулою 2.38.

$$\tilde{t} = med(t). \tag{2.38}$$

Стандартне відхилення часів завершення за формулою 2.39.

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (t_i - \bar{t})^2}. \tag{2.39}$$

По-друге, будуємо лінійну регресію за формулою 2.40.

$$t_{\mathrm{reg}} = \beta_1 w + \beta_0, \tag{2.40}$$

де β_0 – незалежний член;

β_1 – коефіцієнт нахилу.

Умова валідності регресії $p < 0.05$ та $R^2 \geq 0.3$.

По-третє, беремо пропорційну оцінку на основі кількості слів:

Середній час на одне слово за формулою 2.41.

$$\bar{t}_{per\ word} = \frac{\sum t_i}{\sum w_i}. \tag{2.41}$$

Оцінений час за формулою 2.42.

$$t_{\text{words}} = w \cdot \bar{t}_{per\,word}. \tag{2.42}$$

На основі цих трьох методів формуємо агреговану оцінку за формулою 2.43.

$$t_{\text{weighted}} = \frac{\sum w_k \cdot t_k}{\sum w_k}, \tag{2.43}$$

де t_k – оцінки: $\bar{t},\ t_{\text{reg}},\ t_{\text{words}}$;

w_k – ваги методів $0.2, 0.5 * R^2, 0.3$.

Тепер треба провести корекцію на ефективність за часом доби.

Обчислюється ефективність за періодами дня ($\bar{e}_{\text{period}}$), використовуючи формулу 2.44.

$$\bar{e}_{\text{period}} = \frac{1}{n_{\text{period}}} \sum_i \frac{t_i}{w_i}. \tag{2.44}$$

Коригована оцінка ($\bar{e}_{\text{current}}$ – середнє для часу доби створення нової нотатки) обчислюється за формулою 2.45.

$$t_{\text{adj}} = t_{\text{weighted}} \cdot \bar{e}_{\text{current}}. \tag{2.45}$$

Додається запас для заданого рівня довіри α (формула 2.46).

$$t_{\text{rec}} = t_{\text{adj}} + z \cdot s, \tag{2.46}$$

де z – квантиль нормального розподілу для довіри α (або t-розподілу для малих вибірок).

Додатково будуються довірчі інтервали для перевірки (формула 2.47).

$$[t_{\text{adj}} - t_{\text{n}-1} \cdot s\sqrt{1 + \frac{1}{n}};\quad t_{\text{adj}} + t_{\text{n}-1} \cdot s\sqrt{1 + \frac{1}{n}}], \tag{2.47}$$

де $t_{\text{n}-1}$ – критичне значення t-розподілу з $\text{n} - 1$ ступенями свободи.

На останок обчислюємо ймовірність завершення нотатки до дедлайну, використовуючи формулу 2.48.

$$P(completion \le t_{\text{rec}}) = F_t\left(\frac{t_{\text{rec}} - t_{\text{adj}}}{s\sqrt{1 + \frac{1}{n}}}\right), \tag{2.48}$$

де F_t – функція t-розподілу з $\text{n} - 1$ ступенями свободи.

2.3 Архітектурні принципи побудови програми

Архітектура додатку є основою його стабільності, продуктивності та зручності для подальшого масштабування. Вибір правильних принципів проектування забезпечує ефективну взаємодію між компонентами системи, мінімізацію технічного боргу та можливість інтеграції нових функцій без порушення роботи вже існуючих модулів. У цьому підрозділі розглядаються ключові підходи до побудови архітектури, такі як модульність, розділення відповідальності (SOLID-принципи), масштабованість та безпека даних. Використання клієнт-серверної моделі, шарової архітектури (backend, frontend, база даних), а також стандартів мережевої комунікації між компонентами дозволяє створити гнучку та надійну систему. Окрема увага приділяється оптимізації взаємодії з базою даних та інтеграції аналітичних інструментів, що забезпечує високу швидкість обробки запитів і точність аналітики. Ці принципи формують каркас, на якому будується весь функціонал додатку, поєднуючи технічну ефективність із зручністю для кінцевого користувача.

2.3.1 Мотивація розподілу системи

Архітектура системи була розроблена з урахуванням принципів розподілу відповідальностей між окремими підсистемами, що забезпечує чітку спеціалізацію кожного компонента, кращу структурованість проєкту, його масштабованість та зручність подальшого розвитку. Основна ідея полягає у відокремленні взаємодії з користувачем, обробки аналітичних і статистичних даних від процесів зберігання, управління та передачі інформації. Такий підхід дозволяє досягти стійкості до змін: оновлення або вдосконалення однієї з частин,

наприклад, алгоритмів аналітики чи структури бази даних, не вимагає втручання в роботу інших модулів, якщо це не передбачено концептуально.

Кожна компонента системи має чітко визначену зону відповідальності. Фронтенд відповідає за візуалізацію та взаємодію з користувачем, бекенд – за ефективне зберігання, обробку і видачу даних. Модуль аналітики надає персоналізовану статистику, а модуль передбачень формує рекомендації на основі користувацьких даних. Завдяки цьому можлива незалежність розвитку підсистем – наприклад, покращення механізмів статистичного аналізу або заміна бази даних не призводить до потреби змін у фронтенді чи інших компонентах, якщо міжмодульна взаємодія залишається незмінною.

Архітектура також позитивно впливає на продуктивність системи. Частина складних обчислень виконується на стороні клієнта, що дозволяє знизити навантаження на сервер, а бекенд зосереджений на оперативній роботі з базою даних, забезпечуючи швидкий відгук на запити користувача. Це особливо важливо у випадках, коли користувач має велику кількість записів – у таких сценаріях ефективний розподіл обов’язків забезпечує стабільність роботи і швидкодію. Окрім цього, виділення окремих компонент дозволяє точніше контролювати доступ до даних і підвищити рівень безпеки та надійності їх збереження.

Ще однією перевагою є хороша масштабованість: у разі збільшення обсягів даних або навантаження можна масштабувати саме ті частини системи, які цього потребують, наприклад, окремо бекенд або модуль обробки статистики. При цьому фронтенд не вимагає змін, оскільки не має прямої залежності від реалізації внутрішніх механізмів зберігання даних. Завдяки такому розділенню система залишається гнучкою, адаптованою до зростання функціональності й змін технічних вимог без потреби кардинальних змін у загальній структурі.

2.3.2 Розподіл обов'язків між рівнями

Фронтенд рівень (інтерфейс і аналітика) має 3 основні обов'язки.

1. Відповідає за організацію роботи з користувачем: введення, перегляд і редагування нотаток.
2. Виступає як ініціатор запитів до бекенду для збереження або отримання даних. Надає імплементацію транспорту для спілкування з сервером.
3. Надає інтерфейс для запитів до модулю аналітики даних та передбачення і представлення отриманих даних через відповідні елементи зображення для користувачів.

Бекенд рівень (керування даними) має 3 основні обов'язки.

1. Забезпечує централізоване зберігання нотаток та пов'язаних метаданих у базі даних.
2. Відповідає за достовірність і цілісність даних, реалізуючи базові бізнес-правила з перевірки коректності наданих даних.
3. Підтримує простий API для взаємодії з фронтендом, абстрагуючи структуру зберігання від логіки аналізу. Надає імплементацію транспорту для спілкування з клієнтом.

Модуль аналітики даних має 2 основні обов'язки.

1. Описує логіку представлення відповідних аналітичних результатів у вікнах інтерфейсу користувача.
2. Використовує математичні методи для аналітики і прогнозування ефективності виконання нотатки в залежності від дедлайну і часу доби.

Модуль передбачення має 2 основні обов'язки.

1. Використовує каскад методів для передбачення категорії та представлення користувачу декількох альтернатив для вибору найприйнятнішої для нього.

2. Забезпечує створення інтерфейсу користувача у вигляді нового вікна з представленням аналізу категорії, який був виконаний запропонованими методами.

Таким чином, фронтенд оперує високорівневою інформацією, бекенд працює на рівні збереження та базової валідації даних, база даних надає можливість зберігати ці дані локально, модуль аналітики даних проводить дослідження ефективності користувача на базі його дій, а модуль передбачення категорій відповідно рекомендує обрання категорії на базі використання заданих методів. Все це дозволяє зрозуміло розподілити програмні відповідальності між сутностями.

2.3.3 Абстрактна схема взаємодії

Взаємодія компонентів будується за наведеним на рисунку 2.1 описом.

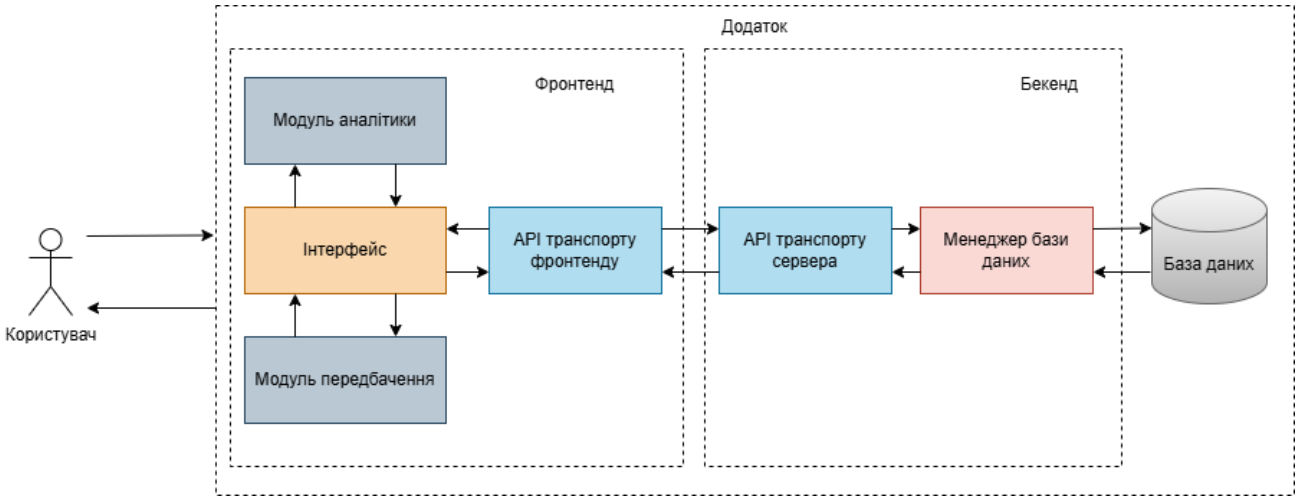


Рисунок 2.1 – Організація програмного продукту.

Наведена вище структура описує типовий алгоритм використання програми. Користувач надсилає свої запити через інтерфейс, який відповідно надає можливість створювати, видаляти та редагувати нотатки. Модулі аналітики і передбачення використовують математичні методи для аналізу на

базі готових нотаток і представляють результат для виведення на інтерфейсі. API транспорту відповідальні за відправку та отримання даних з відповідних сутностей. Менеджер бази даних читає і записує дані з локальної бази. База даних зберігає дані користувача.

2.4 Організація бази даних для збереження нотаток

Для збереження даних у проєкті використовується локальна база даних SQLite. Цей вибір обґрунтований низкою факторів, що поєднують технічні переваги, практичність та відповідність поставленим вимогам цієї роботи. По-перше, обсяг даних є доволі невеликим – нотатки переважно мають лише текстові поля і поля дати, тому використання більш громіздких СУБД на кшталт PostgreSQL чи MySQL є надлишковим. SQLite ідеальний через свою зручність для малих систем: він не вимагає окремого сервера, процесів інсталяції чи адміністрування, а дані зберігаються в єдиному файлі. Це спрощує розгортання, перенесення між пристроями та експлуатацію, оскільки користувач взаємодіє лише з файлом, який автоматично створюється та керується програмою.

Незважаючи на простоту, SQLite забезпечує надійність через підтримку ACID, що критично для запобігання втраті даних при збоях. Транзакційна модель дозволяє безпечно виконувати операції запису чи оновлення, навіть у слабо конкурентних середовищах. Хоча SQLite технічно є реляційною базою, у цьому проєкті він використовується як структуроване сховище з лінійною організацією даних – таблиця notes містить незалежні записи, що близько до моделі key-value або документних баз. Такий підхід уникнув складних зв’язків, зовнішніх ключів або нормалізації, зберігаючи при цьому переваги SQL-інтерфейсу для запитів. Хоча це можна було б вважати приводом для використання NoSQL баз даних, проте у перспективі проєкт може

ускладнюватися новим функціоналом, який вимагатиме перехід до більш складних залежностей у структурі БД.

Важливу роль відіграє крос-платформність SQLite – він підтримується на всіх популярних ОС та інтегрується з найпопулярнішими мовами програмування, що спрощує розробку та підтримку. Продуктивність SQLite для локальних операцій з малими даними вища, ніж у серверних баз, завдяки відсутності мережевих затримок.

Порівняно з альтернативами, SQLite виграє у порівнянні з текстовими файлами, які не забезпечують транзакційної безпеки, ефективних запитів або структурованості, та серверними базами, які надмірні для локального застосунку.

Резервне копіювання реалізується простим копіюванням файлу бази, а експорт даних – стандартними SQL-інструментами. SQLite має доступну документацію, активну спільноту та багаторічну історію стабільної роботи, що зменшує ризики виникнення критичних проблем.

Таким чином, SQLite є оптимальним рішенням для проєкту: він поєднує простоту файлового сховища, структурованість реляційних баз і надійність ACID, не вимагаючи серверної інфраструктури. Це дозволяє зосередитися на логіці застосунку, забезпечуючи користувачам зручність, безпеку та стабільність.

Це лише частина переваг використання бази даних SQLite, наведених з джерела [10].

2.5 Обґрунтування підходу до частини збереження даних

Вибір C++ для реалізації бекенду збереження даних ґрунтується на його здатності забезпечити максимальну продуктивність, надійність та контроль над системними ресурсами, що є вирішальним для локального застосунку з

інтенсивними операціями запису та читання. На відміну від інтерпретованих мов, компільований код C++ дозволяє мінімізувати програмні витрати на обробку даних, що робить його ідеальним для роботи з великими обсягами інформації. Кожен запит до бази даних оптимізується на рівні машинного коду, забезпечуючи миттєве виконання навіть на слабкому обладнанні.

SQLite, інтегрований у C++ через легкі та потужні бібліотеки, виступає ідеальним доповненням. Його архітектура, орієнтована на локальне зберігання у єдиному файлі, усуває необхідність у сервері чи додаткових налаштуваннях, що критично для забезпечення доступності до даних. ACID-властивості SQLite гарантують атомарність транзакцій: якщо користувач додає нотатку, а програма раптово завершується, дані не будуть частково оновлені або пошкоджені. C++ посилює цю надійність: механізми керування пам’яттю (RAII) автоматизують звільнення ресурсів, запобігаючи витокам, а підготовлені SQL-запити захищають від ін’єкцій та прискорюють операції.

Розділення стеку на Python-інтерфейс та C++-бекенд створює чітку архітектурну межу. Python, з його бібліотеками для аналітики, зосереджений на підготовці даних та інтерактивності, тоді як C++ за описом у джерелі [11] виконуватиме «важку» роботу: збереження даних, індексація для швидкого пошуку, транзакційні оновлення.

Гнучкість – ще одна перевага цього підходу. Навіть якщо застосунок розшириться, то C++ дозволить інтегрувати нові функції без змін базової логіки, а використання доступного джерела [12] дозволятиме вирішити будь-які питання щодо імплементації досить швидко.

Порівняно з альтернативами, такими як Python + SQLite або серверні СУБД, зв’язка C++ та SQLite залишається найменш ресурсоємною та найбільш ефективною для локальних застосунків. Вона не вимагає інтернет-з’єднання, не створює навантаження на систему і гарантує, що дані залишаються під повним контролем користувача. Це робить її оптимальним вибором для проєктів, де швидкість, безпека та стабільність є пріоритетами, а інтерфейс лише допомагає взаємодіяти з потужним сервером.

Також можна досягти ще більшої швидкодії сервера, використовуючи наведені в джерелі [13] рекомендації щодо оптимізації коду на мові C++, що в результаті надасть ще більшої переваги використанню саме цієї мови для сервера.

2.6 Обгрунтування підходу до частини відображення даних

Вибір Python з Tkinter для інтерфейсу додатка обумовлений поєднанням технічної практичності, гнучкості мови та специфіки локального користувацького досвіду. Його синтаксис дозволяє швидко розробити прототип логіки взаємодії з даними, а інтеграція бібліотек типу pandas, numpy або scikit-learn надають вже готові рішення для проведення аналізів.

Tkinter, як стандартний GUI-фреймворк, виступає ідеальним доповненням: він не вимагає додаткових залежностей, повністю інтегрований у середовище Python і дозволяє створювати інтерфейс як надбудову над аналітичною логікою, як у джерелі [14].

Незважаючи на візуальну простоту, Tkinter для застосунків із акцентом на функціонал є оптимальним. Його віджети дозволяють відтворити необхідну структуру: список нотаток з фільтрами, графіки продуктивності, поля редагування тощо.

Крос-платформність стеку Python + Tkinter забезпечує роботу додатка на будь-якій ОС без змін у коді, що критично для локального ПЗ. Для користувача це означає відсутність необхідності встановлювати додаткові бібліотеки або налаштовувати середовище – достатньо запустити виконавчий файл або скрипт. На відміну від веб-додатків або важких фреймворків (наприклад, PyQT), цей підхід не створює навантаження на систему, залишаючись ефективним навіть на слабких пристроях.

Таким чином, зв’язка Python + Tkinter – це простий, але потужний інструмент для створення локальних застосунків, де інтерфейс гарно інтегрується з аналітичними процесами. Це дозволяє зосередитись на логіці роботи з даними, а не на складностях архітектури, і залишається найменш ресурсоємним шляхом для реалізації MVP цього типу.

Звертаючись до джерела [15], можна буде у перспективі застосувати більші можливості верстки інтерфейсу до нашої програми з метою масштабування її широкого функціоналу.

2.7 Висновки до другого розділу

Сучасні методи аналізу тексту та передбачення категорій відіграють ключову роль у створенні ефективних застосунків для нотування. Використання аналітики тексту для автоматичної класифікації записів і формування статистики ефективності користувача дозволяє не лише структурувати дані, а й підвищити швидкість планування та аналізу інформації.

Обґрунтований вибір бази даних (SQL, зокрема SQLite) та мов програмування (C++ для бекенду та Python для інтерфейсу) забезпечує оптимальну продуктивність, масштабованість і зручність розробки. Ці технології є основою для реалізації функціоналу, що поєднує швидкість, точність і зручність використання.

Таким чином, застосування сучасних методів аналітики тексту разом із правильно підібраними інструментами розробки створює потужну базу для створення інтелектуального застосунку для нотування. Подібні рішення не просто автоматизують процес організації записів, а й роблять його інструментом для підвищення особистої ефективності.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ НОТАТОК З АВТОМАТИЧНОЮ КАТЕГОРИЗАЦІЄЮ ТА АНАЛІЗОМ ПРОДУКТИВНОСТІ

3.1 Підготовка середовища розробки

Середовищем розробки обрані IDE від JetBrains, а власне CLion для C++ та Pyhcram для Python. Ці додатки надають зручний інтерфейс та інструменти для розробки, дозволяючи як писати власне код, так і налаштовувати його компіляцію/інтерпретацію з можливістю зручно налагоджувати готові бінарні файли.

3.1.1 Середовище розробки серверної частини

Розробка серверної частини ведеться в CLion, оскільки це спеціалізоване середовище для C++, яке ідеально підходить для роботи з низькорівневими та високопродуктивними додатками. Однією з ключових переваг CLion є його тісна інтеграція з CMake, що значно спрощує збірку проекту, керування залежностями та автоматичну генерацію Makefile.

Важливим фактором вибору стала підтримка сучасних стандартів C++, зокрема сімейку (C++17/20), що дозволяє використовувати останні можливості мови для оптимізації продуктивності та чистоти коду. Крім того, CLion надає потужні засоби налагодження, такі як візуальний дебагер з точками зупину, переглядом змінних у реальному часі, покроковим виконанням коду та аналізом пам’яті. Це критично важливо для виявлення складних багів, пов’язаних із керуванням ресурсами в C++.

Додаткова перевага – інтелектуальний аналіз коду, який допомагає знаходити потенційні помилки ще на етапі написання, а також інтеграція з Git, що спрощує роботу в команді та контроль версій.

3.1.2 Середовище розробки користувацької частини

Для клієнтської частини обрано PyCharm – професійне середовище для Python, яке ідеально підходить для швидкої та якісної розробки. Головною перевагою PyCharm є його просунутий дебагер, що дозволяє легко знаходити логічні помилки завдяки можливості покрокового виконання, перегляду змінних і навіть віддаленого налагодження.

PyCharm також надає зручні інструменти для роботи з віртуальними оточеннями, що дозволяє ізолювати залежності проекту (virtualenv, conda, pipenv) та уникнути конфліктів версій. Інтелектуальний редактор із автодоповненням, підказками та автоматичним рефакторингом значно прискорює написання коду, а вбудована підтримка популярних бібліотек (Django, Flask, NumPy, Pandas) робить розробку ще комфортнішою.

Крім того, PyCharm має потужні інструменти для тестування, дозволяючи запускати модульні тести (pytest, unittest) прямо з IDE, що спрощує процес забезпечення якості коду.

3.2 Обрання бібліотек і мови для імплементації

Мови були обрані відповідно до їх переваг і використання у комерційній сфері для схожих задач, C++ вирізняється своєю швидкістю роботи з даними, а Python має широкий вибір бібліотек для швидкої розробки.

3.2.1 Обрання бібліотек і мови для серверу

Для реалізації серверної частини було обрано C++ через його високу продуктивність та ефективне управління пам’яттю.

Обрані бібліотеки, призначення і причини їх обрання для серверу зазначено в таблиці 3.1.

Таблиця 3.1 – Описи обраних бібліотек для серверу.

Бібліотека	*Призначення*	*Чому обрано?*
sys/socket.h	Функціонал для створення мережевої взаємодії (TCP/UDP-сокети).	Забезпечення міжпроцесерної комунікації між фронтендом і бекендом.
nlohmann/json.hpp	Робота з JSON (парсинг, генерація). [16]	Імплементація стандарту для передачі і отримання нотаток у зручній та зрозумілій формі.
sqlite3.h	Робота з локальною базою даних SQLite. [17]	Реалізація методів звернення до БД через програмний інтерфейс.
vector	Стандартна бібліотека для формування динамічних масивів.	Представлення групи нотаток під час роботи програми.
string	Стандартна бібліотека для роботи зі строками.	Робота з текстовими даними нотаток.
stdexcept	Стандартна бібліотека для роботи з виключеннями.	Обробка виключень під час некоректної роботи.

3.2.2 Обрання бібліотек і мови для клієнта

Клієнтська частина реалізована на Python, оскільки ця мова дозволяє швидко розробляти графічні інтерфейси та легко інтегруватися з іншими компонентами системи.

Обрані бібліотеки, призначення і причини їх обрання для клієнта зазначено в таблиці 3.2.

Таблиця 3.2 – Описи обраних бібліотек для клієнта.

Бібліотека	*Призначення*	*Чому обрано?*
tkinter	Представлення програмного інтерфейсу і його графічних підмодулів. [18]	Створення способу взаємодії користувача із функціоналом додатку.
matplotlib	Представлення графіків на базі математичних даних. [19]	Побудова інтерфейсу для представлення деяких результатів модулю аналітики.
socket	Надання функціоналу для створення мережевої взаємодії (TCP/UDP-сокети).	Забезпечення міжпроцесерної комунікації між фронтендом і бекендом.
json	Робота з JSON (парсинг, генерація).	Імплементація стандарту для передачі і отримання нотаток у зручній та зрозумілій формі

Обрані бібліотеки, призначення і причини їх обрання для модулю аналітики зазначено в таблиці 3.3.

Таблиця 3.3 – Описи обраних бібліотек для модулю аналітики.

Бібліотека	*Призначення*	*Чому обрано?*
scipy	Бібліотека для надання готових комплексних математичних функцій. [20]	Використання статистичних методів.
sklearn	Бібліотека для надання готових функцій для аналітики даних і машинного навчання. [21]	Використання логістичної регресії.
numpy	Бібліотека для надання готових базових математичних функцій.	Обчислення квадратів чисел, піднесення до степеня та інше.

Обрані бібліотеки, призначення і причини їх обрання для модулю передбачення зазначено в таблиці 3.4.

Таблиця 3.4 – Описи обраних бібліотек для модулю передбачення.

Бібліотека	*Призначення*	*Чому обрано?*
re	Регулярні вирази для аналізу тексту.	Підготовка тексту перед його обробкою.
Counter	Підрахунок і аналіз частотності тексту.	Спосіб роботи з текстовою статистикою.

3.3 Проєктування та імплементація додатку

Проєктування та впровадження додатку ґрунтується на поєднанні сучасних інженерних підходів із фокусом на потреби користувача. Використання модульної архітектури дозволило розділити систему на незалежні компоненти: інтерфейс (Python), бекенд (C++), базу даних (SQLite) та аналітичний модуль.

Реалізація включає створення логіки для комунікації між серверною та клієнтською частинами, оптимізацію запитів до бази даних для швидкої обробки операцій, а також впровадження інтуїтивного інтерфейсу з візуалізацією статистики (графіки, діаграми). Це створює фундамент для подальшого розширення функціоналу. А використання описаних в джерелі [22] патернів програмування дозволяє підтримувати чистоту коду при розширенні програмного продукту.

3.3.1 Проєктування менеджера бази даних

Суть модулю – проєктування і реалізація класу DatabaseManager, який забезпечує взаємодію з локальною базою даних SQLite. Основна мета – надати централізовану і безпечну роботу з базою, де зберігаються текстові нотатки, а саме: їхній заголовок, вміст, дата створення, категорія, статус, дедлайн і дата завершення.

Клас Note відповідає за абстрактне подання однієї нотатки і містить методи серіалізації (to_json) та десеріалізації (from_json) для зручної роботи з JSON-форматом, що використано в нашій архітектурі у міжпроцесерному обміні даних з клієнтом та при збереженні даних у базу даних. Сам менеджер DatabaseManager у конструкторі приймає шлях до локальної бази даних і викликає метод

init_database, який створює таблицю notes за заданою структурою, якщо вона ще не існує.

Метод get_all_notes виконує SQL-запит до бази й повертає всі нотатки у вигляді вектора об’єктів Note, впорядкованих за датою.

Метод get_note дозволяє отримати конкретну нотатку з бази даних за її переданим ID.

Метод save_note працює як у режимі вставки нової нотатки (insert), так і оновлення вже існуючої (update), залежно від того, чи має нотатка дійсний ID.

Метод delete_note дозволяє видалити нотатку з бази за її переданим ID.

Загалом, клас інкапсулює всю роботу з базою: відкриття, виконання запитів, обробку можливих помилок та закриття з’єднань. Це дозволяє іншим частинам програми працювати з нотатками на високому рівні абстракції, не турбуючись про деталі SQL взаємодії.

3.3.2 Проєктування менеджера транспорту сервера

У цьому фрагменті коду реалізовано компонент SocketServer, який виконує роль серверної частини у клієнт-серверній архітектурі системи нотаток. Абстрактно цей модуль відповідає за мережеву взаємодію – тобто приймання підключень від клієнтів через TCP-сокети, обробку запитів у форматі JSON, взаємодію з базою даних через DatabaseManager і повернення результатів клієнту. Основна ціль – забезпечити зовнішній інтерфейс до функціоналу збереження, отримання, видалення та переліку нотаток у віддаленому режимі.

Конструктор класу ініціалізує серверний сокет на заданому порту та зберігає посилання на об’єкт бази даних.

Метод setup_socket створює сокет, налаштовує його на повторне використання адреси, прив'язує до IP-адреси та порту, і ставить у режим очікування підключень.

Головний цикл обробки запитів реалізовано у методі run_server_loop – він постійно перевіряє (через select) наявність активності як на сокеті сервера, так і на вже встановленому з клієнтом з'єднанні, приймає нові підключення, та викликає process_client_data для обробки отриманих даних.

Метод process_client_data читає буфер вхідних повідомлень від клієнта і накопичує їх до повного повідомлення (відокремленого символом нового рядка).

Коли повідомлення повне – воно обробляється методом handle_message. Саме тут відбувається розбір JSON-структури запиту, визначення запитаної операції (за реалізованими в програмі: «save», «get», «delete», «list») та відповідне делегування виклику методів об’єкта бази даних. Кожна операція завершується формуванням JSON-відповіді, яка надсилається назад клієнту методом send_response.

Загалом, цей клас реалізує повноцінний TCP-сервер, який дозволяє клієнту звертатися до бази даних нотаток, використовуючи простий протокол передачі повідомлень у форматі JSON. Такий підхід робить систему масштабованою та придатною до інтеграції з іншими програмами чи інтерфейсами, наприклад, мобільними або вебклієнтами.

Сам сервер запускається у головному main файлі з однойменною функцією. Якщо треба задати відповідний порт чи базу даних, які відрізняються від автоматично заданих програмою, можна використовувати відповідні прапорці (--port PORT та --db PATH відповідно) для бінарного файлу з метою задання власних налаштувань. Цикл у main існує весь час до термінації програми викликом SIGINT або SIGTERM.

3.3.3 Проєктування менеджера транспорту клієнта

Модуль називається NoteSocketService та реалізує клієнтську частину системи нотаток, яка взаємодіє з C++ сервером через TCP-сокети. Абстрактно,

цей клас є проксі-шаром між GUI-компонентом користувача і бекендом на C++, надаючи засоби для надсилання запитів (збереження, отримання, видалення, отримання списку нотаток) і приймання відповідей у форматі JSON. Це дозволяє Python-додатку працювати з C++-бекендом так, ніби це локальний сервіс, приховуючи низькорівневі деталі міжпроцесерної комунікації.

Конструктор ініціалізує параметри підключення, створює сокет і готує обробку повідомлень у фоновому потоці. Метод connect встановлює TCP-з’єднання з сервером і запускає фоновий потік _receive_messages, який постійно слухає вхідні повідомлення. У разі втрати з'єднання реалізовано просту логіку повторного підключення через метод _ensure_connection.

Методи send_note, delete_note, get_note та request_notes формують відповідні JSON-запити для операцій з нотатками й відправляють їх серверу. Дані передаються у форматі JSON, причому повідомлення завершуються символом нового рядка, що слугує маркером кінця повідомлення для розбору на сервері.

Метод set_receive_callback дозволяє встановити функцію зворотного виклику, яка виконується щоразу, коли клієнт отримує повідомлення від сервера. Потік _receive_messages виконує зчитування з сокета, накопичує вхідні дані, ділить їх на повні JSON-повідомлення та викликає callback, якщо його задано. Це дозволяє асинхронно обробляти відповіді від сервера без блокування основного інтерфейсу.

Таким чином, NoteSocketService виступає посередником між інтерфейсом користувача та логікою зберігання даних, реалізованою на C++. Він абстрагує всі технічні деталі мережевого з’єднання і забезпечує стабільну двосторонню комунікацію з C++ сервером, дозволяючи Python-клієнту працювати з нотатками як із віддаленим сервісом.

3.3.4 Проєктування модулю аналітики нотаток

Реалізовано клас NotesAnalytics, який виступає ідейним центром функціональної підсистеми інтелектуального аналізу користувацьких нотаток. Основна мета даного модуля полягає у виявленні статистичних закономірностей у поведінці користувача, аналізі продуктивності, змістовному розборі нотаток та генерації персоналізованих висновків.

Метод set_notes дозволяє завантажити колекцію нотаток, з якою працюватимуть всі подальші методи. Далі, для збору базових метрик застосовується метод get_summary_statistics, який підраховує загальну кількість нотаток, їх розподіл за статусами («todo», «done»), категоріями, а також обчислює середній час виконання і частку вчасно завершених завдань.

Для детального аналізу за категоріями передбачено метод get_category_stats, який групує нотатки за категоріями, обчислює частки статусів у кожній групі, середній, медіанний та стандартний відхил у часі виконання, а також дотримання дедлайнів.

Аналіз швидкості виконання завдань реалізовано через метод get_completion_speed. Він виконує регресійний аналіз залежності між кількістю слів у нотатці та часом виконання (з використанням stats.linregress), визначає найефективніші категорії (у перерахунку годин на слово) та час доби, в який користувач працює найпродуктивніше. При цьому використовується також дисперсійний аналіз (ANOVA), щоб виявити, чи різниця у продуктивності між періодами доби статистично значуща.

Метод get_deadline_compliance зосереджений на аналізі дотримання дедлайнів. Він оцінює середню затримку або випередження в здачі нотаток, підраховує частку завдань, виконаних вчасно, а також виконує кореляційний аналіз між розміром буфера часу (час між створенням і дедлайном) і ймовірністю завершення завдання. Тут також застосовується логістична регресія

(LogisticRegression з sklearn) для побудови моделі передбачення на основі буфера часу.

Аналіз змісту реалізовано у методі get_content_analysis. Він визначає середню довжину тексту (у словах), оцінює розподіл довжини по категоріях, виявляє зв’язок між довжиною тексту та часом виконання через регресійний аналіз, а також виконує лексичний розбір. Останній включає підрахунок найпопулярніших слів у кожній категорії та обчислення TF-IDF значень для виявлення найбільш характерної лексики.

Однією з ключових можливостей модуля є генерація рекомендацій, реалізована через метод get_actionable_insights. На основі зібраної статистики він формує текстові висновки, наприклад, яка категорія є найефективнішою, який час доби – найпродуктивніший, чи має буфер часу позитивний вплив на завершення задач. Ці висновки представлені в структурованому вигляді й можуть бути виведені в інтерфейсі.

Окремо варто виділити функцію get_deadline_recommendation, яка генерує персоналізовану оцінку тривалості виконання нової нотатки з урахуванням її довжини, історичних даних по вибраній категорії, часу доби та інших факторів. Вона застосовує кілька методів оцінювання – середнє значення, регресійну модель, пропорційне прогнозування по словах – і обчислює зважене значення для прогнозу. Окрім цього, формується часовий інтервал із різними рівнями довіри, а також оцінюється ймовірність вчасного завершення завдання.

Таким чином, проєктування модуля NotesAnalytics реалізоване як комплексна система статистичного й поведінкового аналізу, що включає класичні методи статистики (середні, стандартні відхили, кореляція, регресія), часові патерни (аналіз добового ритму), змістовну обробку (TF-IDF, частотний аналіз), а також механізми генерації висновків. Така архітектура дозволяє інтегрувати аналітичну підсистему в інтерфейс користувача для підвищення ефективності персональної організації задач.

3.3.5 Проєктування модулю передбачення категорій

Цей модуль відіграє важливу роль у підвищенні зручності користувача, оскільки дозволяє системі самостійно віднести нову нотатку до відповідної категорії без ручного уведення. У проєкті реалізація цієї функціональності здійснена через класичні та семантичні алгоритми на основі текстової обробки природної мови.

Модуль реалізовано як набір класів, що наслідують спільну базову структуру BaseCategoryPredictor, яка відповідає за попередню обробку тексту. У цьому базовому класі реалізовано метод preprocess_text, який приводить текст до нижнього регістру, видаляє небуквені символи, цифри, а також фільтрує поширені стоп-слова, тим самим готуючи текст до подальшого аналізу. Також передбачено функцію підрахунку частотності слів – get_word_frequencies.

Кожна модель класифікації є окремим підкласом, який реалізує власну стратегію прогнозування через метод predict_category. Зокрема, клас TFIDFCosinePredictor використовує класичний підхід TF-IDF у поєднанні з косинусною подібністю між вектором нової нотатки та зведеним вектором кожної категорії. Розрахунок проводиться через функції calculate_tf_idf та cosine_similarity, що дозволяє оцінити близькість між семантичним вмістом документів.

Клас NaiveBayesPredictor реалізує наївний байєсівський класифікатор. Під час навчання в методі train враховується частота слів у кожній категорії та загальні кількості документів, а під час передбачення застосовується логарифмічна обробка ймовірностей із використанням згладжування Лапласа.

Ще одна модель – JaccardSimilarityPredictor, яка використовує коефіцієнт Жаккара для порівняння множин слів між нотаткою та категоріями.

Далі – FrequencyDistancePredictor, який визначає схожість на основі порівняння нормалізованих частотних векторів слів з використанням різних

метрик відстані (евклідова, мангеттенська, або чебишевська). Метод calculate_distance дозволяє гнучко налаштовувати спосіб обчислення подібності.

Найскладніший і найсучасніший підхід реалізовано у класі LSAPredictor. Ця модель поєднує TF-IDF із латентно-семантичним аналізом (LSA), що базується на сингулярному розкладі матриці (SVD). Метод compute_tf_idf_matrix будує матрицю TF-IDF, truncated_svd проводить розклад, а document_vector проектує нову нотатку в семантичний простір. Після цього здійснюється порівняння з векторами середніх представлень кожної категорії в просторі LSA через косинусну подібність.

Для зручності інтеграції в систему всі моделі об’єднано в клас MultiMethodCategoryPredictor, який інкапсулює всі предиктори та дозволяє запускати прогнозування одночасно кількома методами. Його метод predict_category повертає набір результатів для кожного підходу, а метод get_consensus_prediction дозволяє обрати найпопулярнішу відповідь серед моделей як фінальний результат.

Таким чином, проєктування модуля прогнозування категорії нотаток здійснено з урахуванням порівняння кількох підходів. Така архітектура дозволяє не лише досягти високої точності класифікації, а й забезпечити гнучкість, масштабованість та можливість розширення або заміни окремих компонентів без зміни загальної структури системи.

3.3.6 Проєктування інтерфейсу

Інтерфейс програми реалізований через систему взаємопов'язаних класів, кожен з яких відповідає за певний аспект роботи з нотатками.

Головний клас NoteflowUI успадковує функціонал PredictorUI та керує комунікацією з сервером через сокети. Він відповідає за всі операції з нотатками: створення нових, збереження змін, видалення та синхронізацію з сервером. Коли

користувач додає або редагує нотатку, цей клас формує відповідні запити до сервера, а потім обробляє отримані відповіді, оновлюючи інтерфейс.

Базовий функціонал інтерфейсу реалізований у класі NotepadInterface. Він створює основні елементи GUI: список нотаток з можливістю пошуку та фільтрації, поля для редагування тексту, вибору категорій та статусів. Тут же реалізована логіка відображення обраної нотатки, зміни її статусу та управління дедлайнами.

Клас PredictorUI розширює базовий інтерфейс, додаючи інтелектуальні функції. Він використовує алгоритми аналітики тексту для передбачення категорій нотаток на основі їхнього змісту. Коли користувач натискає кнопку «Передбачити категорію», програма аналізує текст за допомогою кількох методів одночасно і пропонує найбільш ймовірний варіант. Також цей клас відповідає за рекомендації дедлайнів – він аналізує історичні дані про час виконання подібних нотаток, враховує поточний час доби та інші фактори, щоб запропонувати оптимальний термін виконання.

Окремий клас AnalyzerUI забезпечує візуалізацію всієї аналітики. Він генерує звіти та графіки, які допомагають користувачеві зрозуміти свої звички: які категорії нотаток виконуються швидше, у який час доби працюється найпродуктивніше, як часто вдається вкладатися в дедлайни.

Усі ці компоненти працюють разом, щоб забезпечити зручний інтерфейс для роботи з нотатками, який не лише дозволяє їх створювати та редагувати, але й надає корисні інтелектуальні підказки на основі аналізу даних.

Додатково для додатка використано іконку, яку зображено на рисунку 3.1, з однією з запропонованих назв застосунку. Вона була створена з-за допомоги інструментів джерела [23].

Рисунок 3.1 – Логотип і прототипне ім’я додатку.

3.3.7 Спосіб збірки програми

Збірка програми здійснюється у кілька етапів, оскільки вона складається з двох частин – бекенду, написаного мовою C++, та фронтенду, реалізованого на Python.

Перш за все виконується збірка бекендової частини. Для цього використовується система керування збіркою CMake, використання якого детально описано в [24], а також компілятор C++ і менеджер пакетів. Вихідні коди зберігаються у відповідних теках. У CMakeLists.txt задаються основні параметри проєкту, включаючи стандарт мови програмування, шляхи до файлів та зовнішні залежності (бібліотека SQLite, JSON). Після налаштування проєкту в окремій теці build/ виконується генерація збірних файлів, після чого здійснюється безпосередня компіляція. У результаті генерується бінарний виконавчий файл, який функціонує як окремий сервіс.

Фронтенд програми побудований на Python. Основний файл main.py, що міститься в модулі Frontend, є точкою входу і координує роботу всіх інших підмодулів програми. Всі ці модулі реалізують окремі частини інтерфейсу або логіки. Для запуску використовується командний виклик «python -m Frontend.main».

Для запуску фронтенду без встановлення Python на кінцевому комп’ютері, він може бути зібраний у самостійний виконуваний файл. Для цього застосовується утиліта PyInstaller, яка перетворює main.py у .exe (на Windows) або відповідний виконуваний файл на інших платформах.

У результаті створюється єдиний файл, який користувач може запускати напряму. Обидві сутності сервера і клієнта в результаті є незалежними і можуть запускатися в зручних для користувача місцях.

Таким чином, результатом є повноцінна крос-платформна програма, яка поєднує ефективність C++ для обробки даних і зручність Python для побудови користувацького інтерфейсу, при цьому підтримуючи можливість простого запуску як під час розробки, так і після збирання.

3.4 Наведення прикладів використання продукту

Додаток має широкий спектр ситуацій використання і можливих результатів, тому для розуміння його використання є доцільним зобразити один з варіантів взаємодії з готовим продуктом. Найкращим прикладом для демонстрації цього буде наведені зображення базового функціоналу частин клієнта та сервера за звичних умов функціонування.

3.4.1 Результати роботи серверу

Сервер працює виключно в режимі консолі і надає переважно інформацію про стан комунікації і надає логи під час його роботи.

Для початку запустимо сервер, його ініційований стан зображено на рисунку 3.2.

```
andr1ano@DESKTOP-UBJJSRU:/mnt/c/Users/andr1ano/Documents/KPI/ДИПЛОМ/NoteflowAI/Backend$ ./socket_handler
Starting Notes Server...
Database: note_db/notes.db
Port: 8080
Database initialized successfully
Socket server set up on port 8080
Waiting for connections...
```

Рисунок 3.2 – Старт сервера.

Сервер реагує відповідно до отриманих команд, і усі запроси і відповіді виводить на ту саму консоль. На рисунку 3.3 зображено прийняття нового підключення від клієнта.

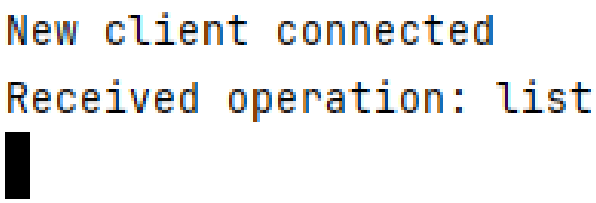


Рисунок 3.3 – Підключення нового клієнта.

Як бачимо, тут сервер при підключенні клієнта отримав відразу запит на отримання списку нотаток, збережених у базі даних. Наведемо приклад комунікації з клієнтом, викликавши зі сторони інтерфейсу запит на збереження нової нотатки. Результат цього запиту зображено на рисунку 3.4.

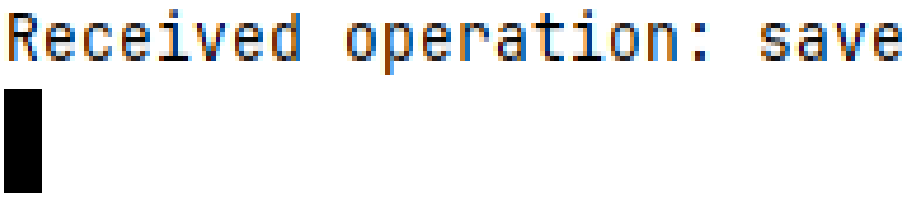


Рисунок 3.4 – Результат сервера на запит збереження нової нотатки.

Працюючи за однаковим інтерфейсом, сервер буде логувати усі запити клієнта однаковим чином.

В ситуації, коли клієнт відключився від сервера, про це також буде створений лог сервером. Приклад такого логу зображено на рисунку 3.5.

Рисунок 3.5 – Інформація про відключення клієнта.

Сервер можна вбити, викликавши команди SIGINT або SIGTERM. Результатом цього буде вимкнення сервера і відповідно зупинка ним роботи з обробки запитів клієнта. Для поновлення роботи потребується знову запускати виконавчий файл сервера. Результат термінації сервера представлено на рисунку 3.6.

```
^C
Received signal 2. Shutting down server...
andr1ano@DESKTOP-UBJJSRU:/mnt/c/Users/andr1ano/Documents/KPI/ДИПЛОМ/NoteflowAI/Backend$
```

Рисунок 3.6 – Ліквідація сервера.

3.4.2 Результати роботи клієнта

Клієнт надає більш ширший функціонал і можливість для взаємодії. Початковим його елементом є власне головний інтерфейс (рисунок 3.7.).

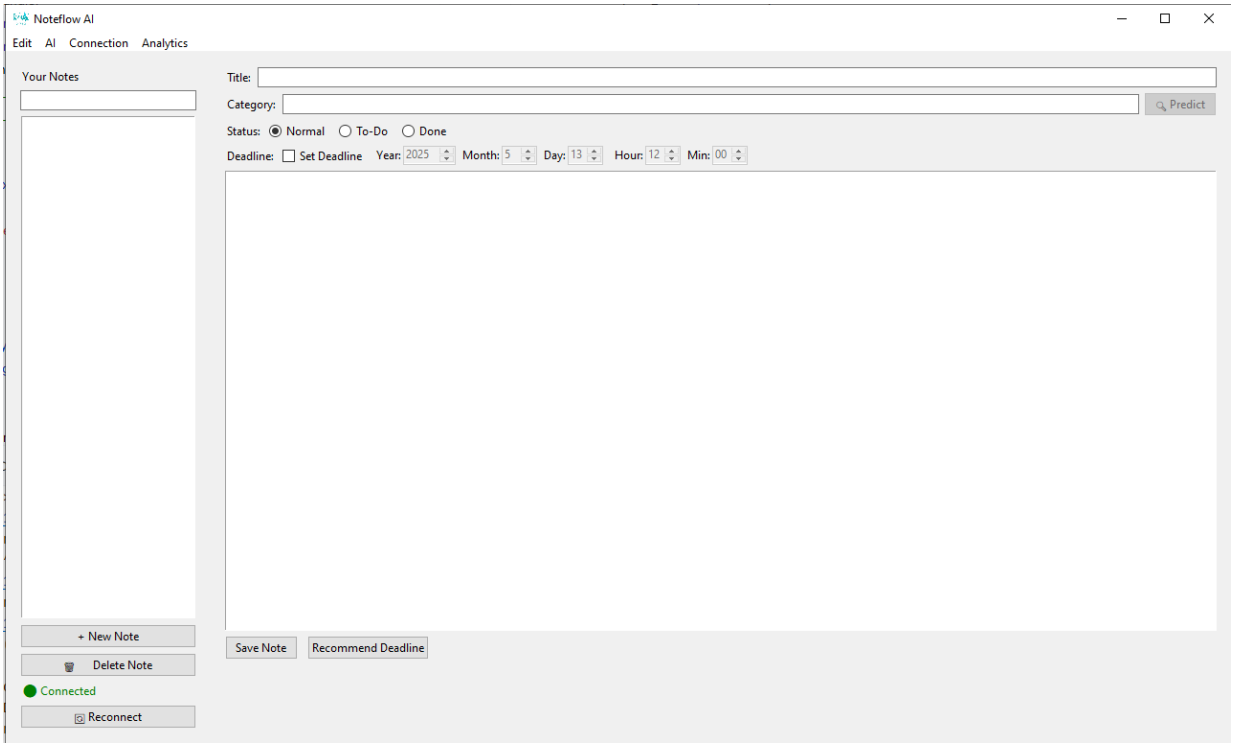


Рисунок 3.7 – Представлення головного інтерфейсу.

Базовий функціонал передбачає створення і збереження нової нотатки. Для цього користувач використовує відповідні елементи наданого функціоналу. Наприклад на рисунку 3.8 зображено варіант створення нової нотатки.

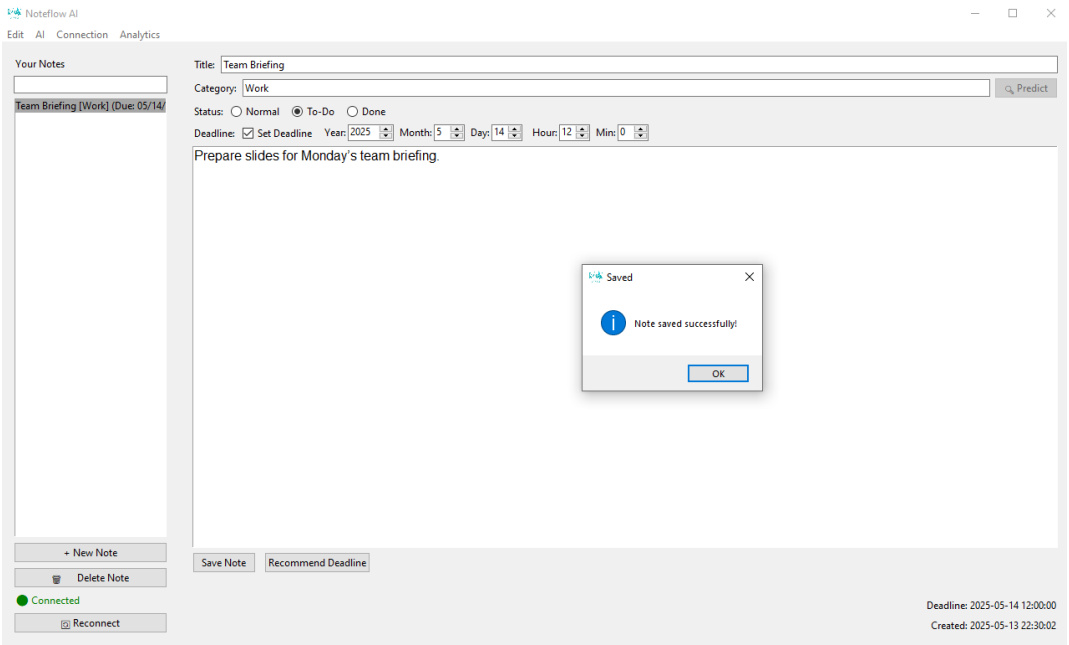


Рисунок 3.8 – Створення нової нотатки.

Відповідним чином користувач може створювати і видаляти такі нотатки, використовуючи надані елементи програми.

Розглянемо тепер модуль передбачення, для цього підготуємо нашу базу даних, аби інтерфейс мав вигляд як на рисунку 3.9.

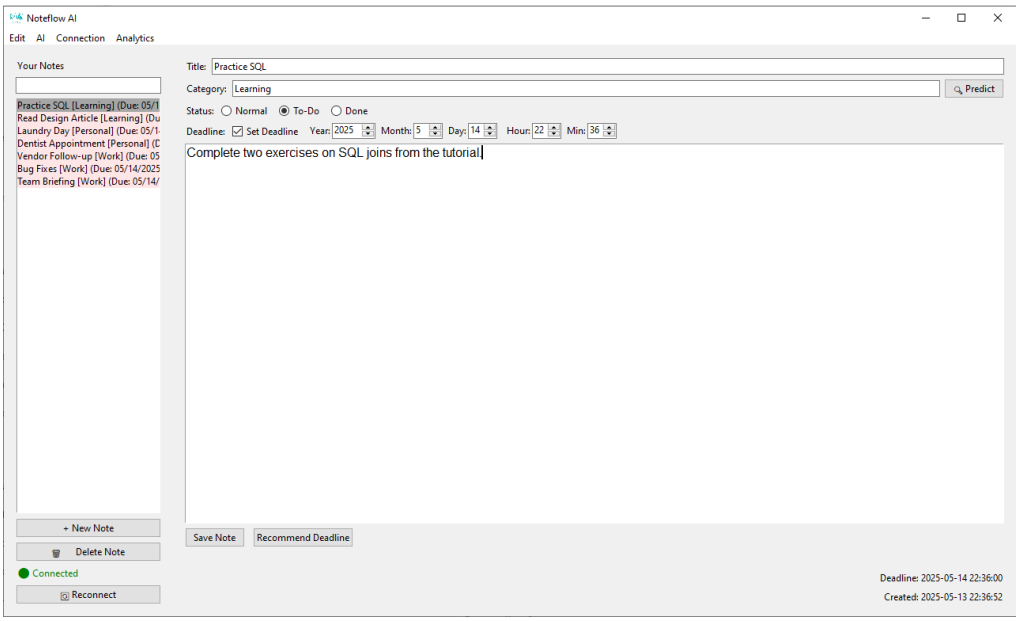


Рисунок 3.9 – Підготовлений інтерфейс для використання передбачення.

Було підготовлено 8 нотаток (3 з категорії «Work», 2 з категорії «Personal» і 2 з категорії «Learning»). Для тестування передбачення створюємо нову нотатку без категорії з назвою «Team Check-In» і текстом «Confirm tomorrow’s team sync and prepare discussion points.». Вважатимемо, що найточніша для неї категорія є «Work». Перевірка цього припущення програмою наведена на рисунку 3.10.

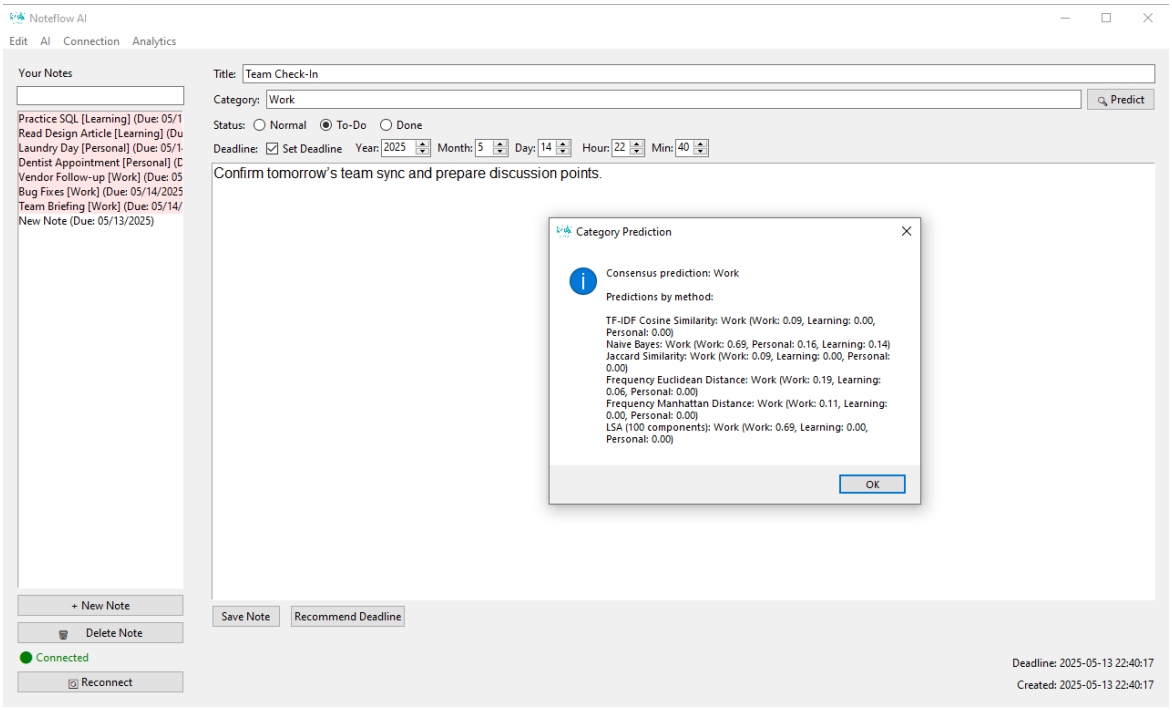


Рисунок 3.10 – Знаходження категорії для нової нотатки.

Як бачимо, голосування предикторів також визначила категорію нотатки як «Work», що дозволяє верифікувати роботу цього модулю.

На останок треба розглянути роботу модулю аналітики, для цього підготуємо наші нотатки з точки зору встановлених дат створення, дедлайнів і дат завершення, аби воно мало вигляд як на рисунку 3.11.

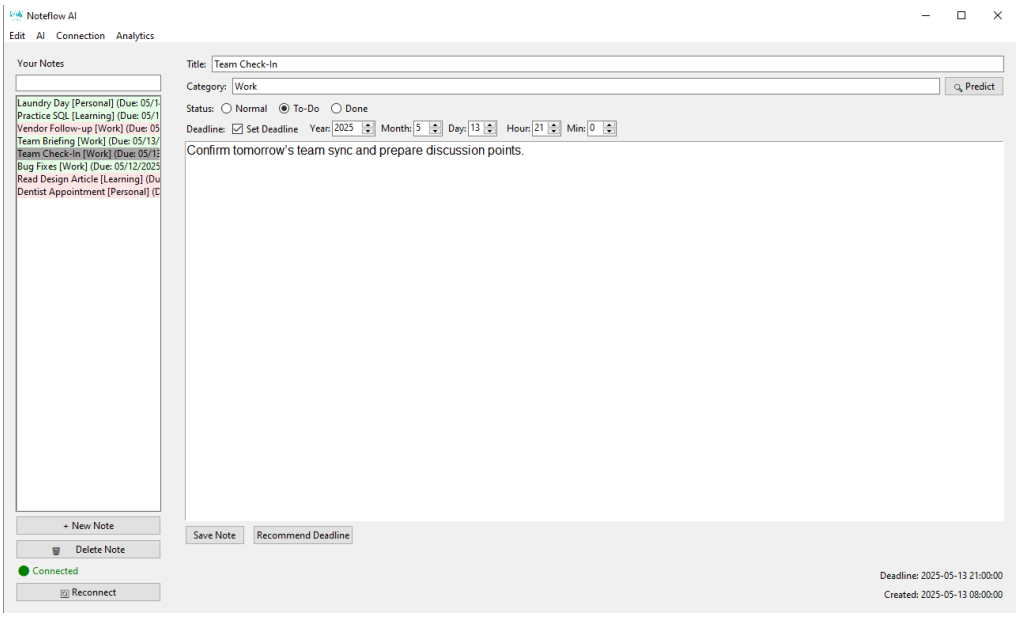


Рисунок 3.11 – Підготовлені нотатки для модуля аналітики.

Використаємо для прикладу агрегуючий функціонал «Get Actionable Insights». Він надає основну статистику по головним параметрам роботи користувача з програмою. Результати обчислень наведено на рисунку 3.12.

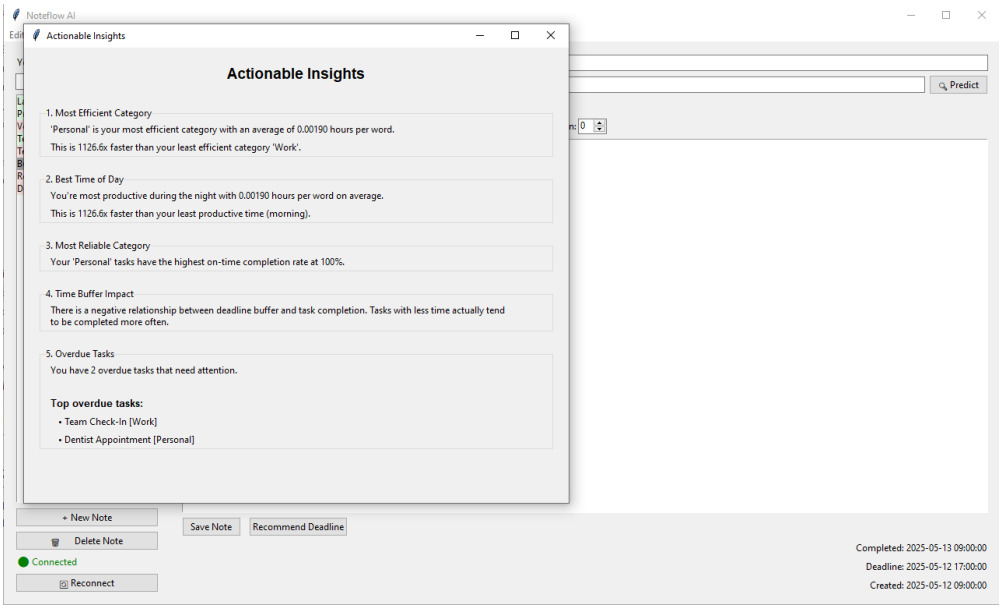


Рисунок 3.12 – Результати виклику Get Actionable Insights.

Також наведемо на рисунку 3.13 результат рекомендованого дедлайну для нашої нотатки «Team Check-In».

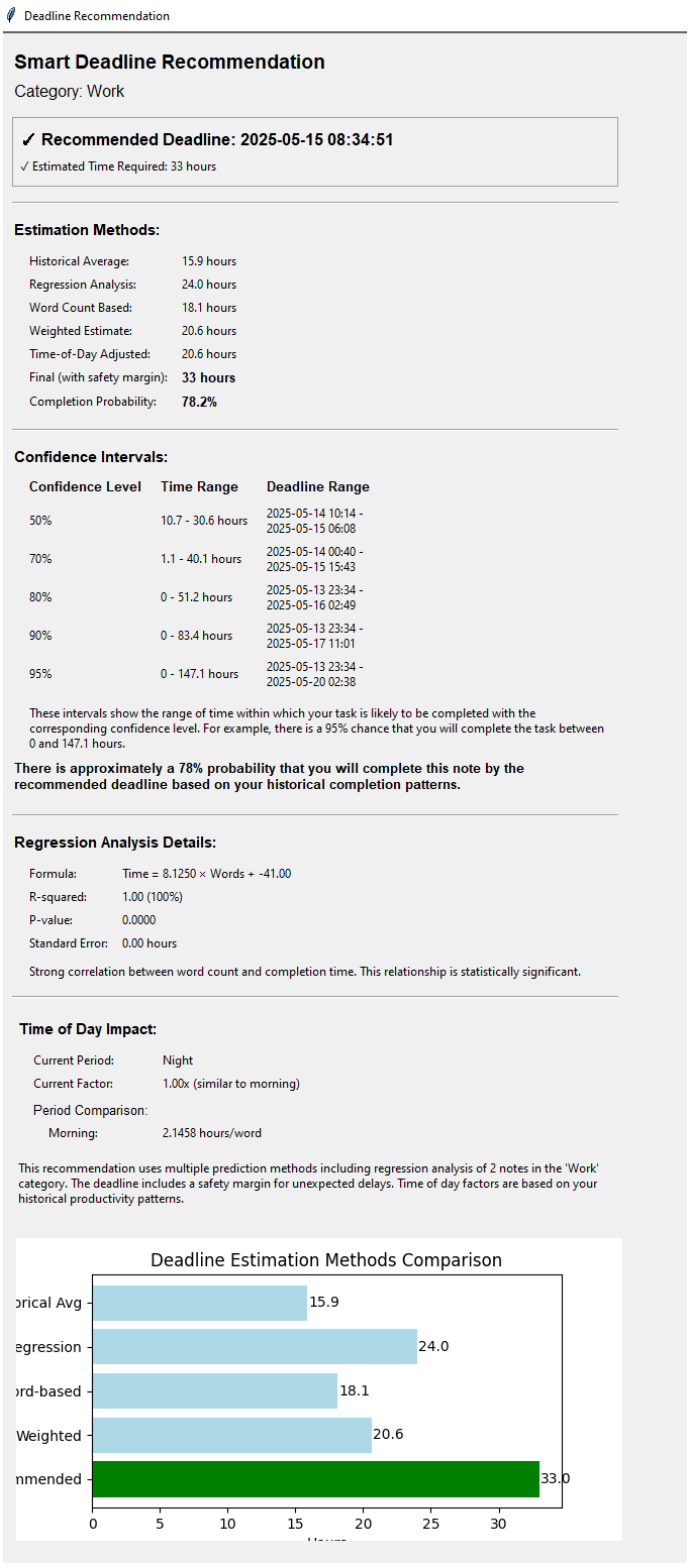


Рисунок 3.13 – Результат обчислення рекомендованого дедлайну.

В результаті маємо зручний додаток, який надає потужний аналітичний модуль і зручний інтерфейс для отримання обчислень власної ефективності.

3.5 Висновки до третього розділу

Реалізація додатку для нотувань із автоматичною категоризацією та аналізом продуктивності продемонструвала ефективність застосування сучасних технологій і методів, описаних у попередніх розділах. Інтеграція моделей аналітики тексту для класифікації змісту, розробка інтуїтивного інтерфейсу (на базі Python) та високопродуктивного бекенду (на C++) забезпечили стабільну роботу системи, швидкість обробки даних і точність аналітичних звітів.

Використання SQLite як основи для зберігання структурованих даних довело свою гнучкість і надійність, дозволивши ефективно керувати великими обсягами інформації. Автоматизація категоризації записів на основі текстових алгоритмів, а також генерація статистики продуктивності (часові графіки, побудова регресії, аналіз ключових слів) значно спростили взаємодію користувача із системою.

Таким чином, успішна реалізація додатку підтвердила життєздатність обраного підходу. Поєднання автоматизації, аналітики та зручного інтерфейсу перетворює програму не лише на інструмент для нотувань, а й на платформу для осмисленого планування, моніторингу та покращення особистої ефективності. Це створює передумови для подальшого вдосконалення системи шляхом додавання нових функцій та масштабування під потреби різних користувацьких аудиторій.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі аналізуються ключові характеристики програми для підвищення персональної продуктивності, яка використовує автоматизоване ведення записів із подальшим аналізом ефективності. Основне завдання дослідження полягає у розробці програмного рішення, здатного забезпечити зручне керування персональними даними, автоматичне сортування інформації за категоріями та формування звітів для оцінки продуктивності. Такий інструмент має широку сферу застосування – від індивідуального використання до професійного впровадження в області управління часом, освітніх процесів та керування проектами.

Під час дослідження розглядались різні підходи до впровадження системи, включаючи застосування функціонально-вартісного аналізу для вибору найефективнішого методу обробки інформації та досягнення оптимального співвідношення витрат і якості. Цей аналіз дає змогу точно визначити вартість кожного функціоналу програми, такого як розпізнавання текстів, інтеграція з іншими платформами та наочне представлення статистичних даних. Він дозволяє знаходити шляхи зниження витрат без погіршення якості, забезпечуючи ідеальний баланс між корисністю для кінцевого користувача та собівартістю продукту.

Результати дослідження свідчать, що продуктивність застосунку безпосередньо залежить від точності текстових алгоритмів та зручності інтерфейсу. Система демонструє високу швидкодію при роботі з великими масивами даних, здатність адаптувати категорії під конкретні потреби користувача та точність формування аналітичних відомостей. Оптимальна модель, яка поєднує в собі швидкість обробки, точність класифікації та гнучкість налаштувань, підтверджує ефективність запропонованого рішення для покращення особистої продуктивності через автоматизований аналіз записів.

4.1 Постановка задачі проєктування

Дослідження використовує метод функціонально-вартісного аналізу (ФВА) для проведення техніко-економічної оцінки системи підтримки особистої ефективності на основі автоматизованого ведення нотаток. Фактичний аналіз зосереджений на оцінці функцій програмного забезпечення, призначеного для збору, категоризації та аналізу персональних даних з метою покращення продуктивності користувача.

Технічні вимоги до програмного продукту включають чотири аспекти.

1. Програмне рішення має функціонувати на звичайних персональних комп'ютерах та мобільних пристроях без необхідності спеціального обладнання. Воно повинно підтримувати популярні операційні системи (Windows, macOS, Linux, Android, iOS) та працювати на типових апаратних конфігураціях.
2. Система повинна мати простий і зрозумілий інтерфейс, що дозволяє користувачам швидко освоїти основний функціонал – від створення нотаток до перегляду аналітики. Важливим елементом є адаптивність інтерфейсу під індивідуальні уподобання (темна/світла тема, налаштування категорій тощо).
3. Застосунок має забезпечувати миттєву обробку текстових даних, автоматичну категоризацію з використанням NLP-алгоритмів та генерацію звітів без затримок. Користувач повинен отримувати актуальну статистику своєї продуктивності у реальному часі, що особливо важливо для оперативного планування завдань.
4. Система повинна бути розроблена з урахуванням майбутнього розширення функціоналу – наприклад, додавання інтеграції з іншими сервісами (Google Calendar, Trello, Notion) або вдосконалення

аналітичних інструментів. Архітектура має дозволяти легко вносити зміни, виправляти помилки та оновлювати алгоритми без серйозних переробок коду.

А також 3 додаткових критерії.

1. Безпека даних – захист персональної інформації користувачів через шифрування та надійні методи автентифікації.
2. Офлайн-робота – можливість ведення нотаток без інтернет-з’єднання з подальшою синхронізацією.
3. Хмарна інтеграція – автоматичне збереження даних у хмарних сховищах для доступу з різних пристроїв.

Функціонально-вартісний аналіз дозволяє визначити оптимальний баланс між вартістю розробки, технічними можливостями та користю для кінцевого користувача, що є критично важливим для створення конкурентоздатного продукту на ринку додатків для продуктивності.

4.2 Обґрунтування функцій програмного продукту

Продукт є комплексним та багатовекторним у своїй реалізації, тому треба виконати певну функціональну декомпозицію його. Головна функція (F0) – розробка програмного продукту для створення зберігання, автоматичної категоризації та надання аналітики нотаток. На основі цієї функції виділено три підфункції.

1. F1 – вибір мови програмування для бекенда;
2. F2 – вибір мови програмування для фронтенда;
3. F3 – вибір основної математичної бібліотеки.

Кожна функція має декілька варіантів реалізації, що відповідають її специфіці.

Функція F1:

1) мова програмування Java;
2) мова програмування C++.

Функція F2:

1) мова програмування Python;
2) мова програмування C++.

Функці F3:

1) бібліотека SciPy;
2) бібліотека Sklearn.

Морфологічна карта (Рисунок 4.1) відображає всі можливі комбінації реалізації цих функцій.

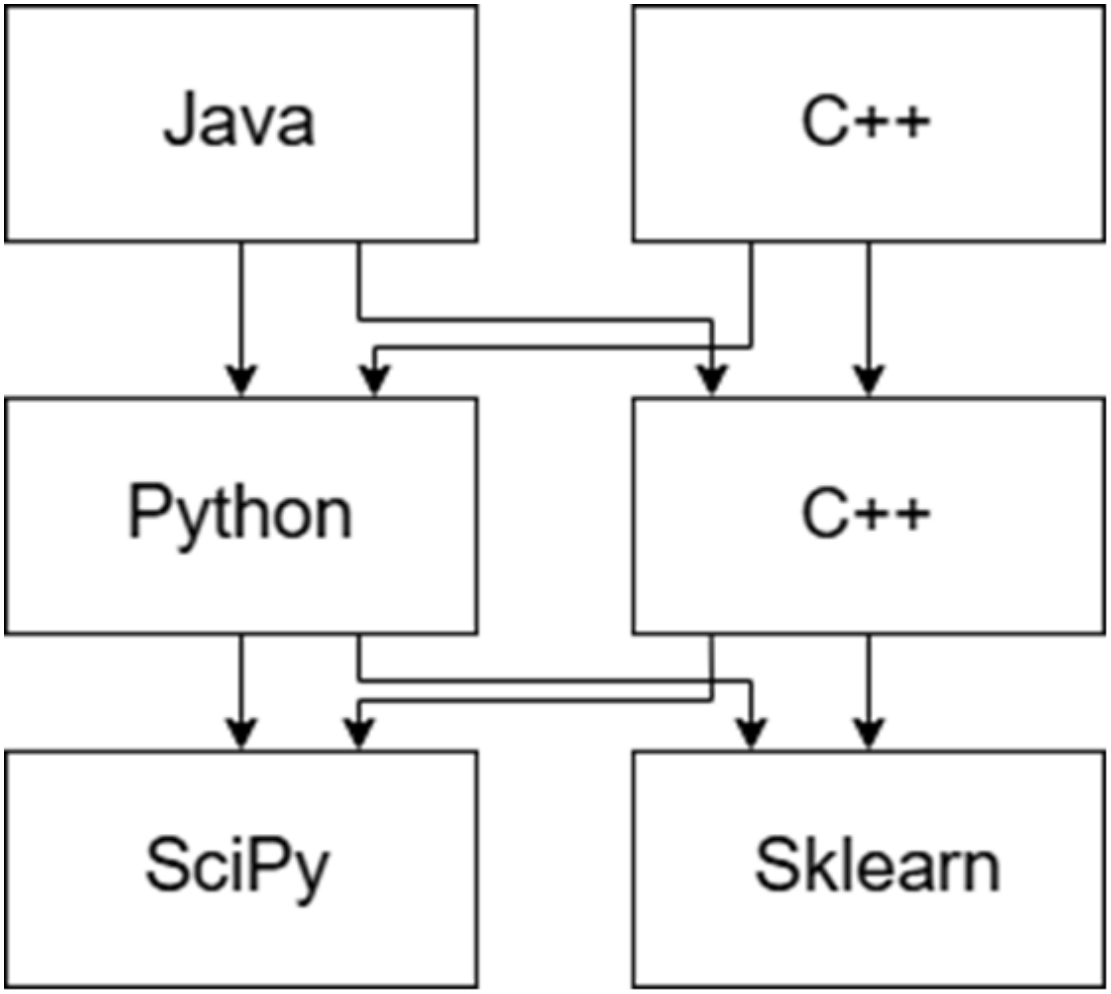


Рисунок 4.1 – Морфологічна карта

Зобразивши нашу структуру, можна перейти до формування позитивно-негативної матриці (таблиця 4.1).

Таблиця 4.1 – Позитивно-негативна матриця.

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Крос-платформність, велика кількість бібліотек, хороша підтримка багатопоточності.	Повільніша за C++ у обчислювально важких задачах, вища споживання пам'яті.
	Б	Висока швидкодія, низькорівневий контроль, ефективне використання пам'яті.	Складніший синтаксис, менш безпечний у порівнянні з Java.
F_2	А	Простота використання, велика кількість бібліотек, швидкий розвиток проектів.	Повільний у обчисленнях, GIL обмежує багатопотоковість.
	Б	Висока продуктивність, підходить для системного програмування.	Складність написання коду, менш придатний для швидкого прототипування.
F_3	А	Оптимізовані математичні алгоритми, підтримка наукових обчислень.	Менш зручний для ML у порівнянні з Scikit-learn, складніший API.
	Б	Зручний інтерфейс для ML, інтеграція з NumPy/Pandas, хороша документація.	Обмежена підтримка глибокого навчання, менш ефективний для великих даних.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не повністю задовольняють поставлені перед програмним продуктом задачі. Отже робимо відповідні висновки про функції.

Функція F1: Обрано C++, відкинуто Java. C++ вибрано через високу швидкодію та ефективне управління пам'яттю, що критично для продуктивності програми. Java відхилено через надмірне споживання ресурсів та нижчу швидкодію в обчислювальних задачах.

Функція F2: Обрано Python, відкинуто C++. Python обраний через простоту розробки, велику кількість бібліотек для інтерфейсу та швидкий прототипування. C++ відкинуто через складність створення GUI та відсутність необхідності у високій продуктивності на фронтенді.

Функція F3: Допустимі обидва варіанти – Sklearn або SciPy. Sklearn підходить, якщо більше потрібні функції машинного навчання, а SciPy – для більш наукових обчислень. Вибір залежить від того, який саме математичний апарат нам буде необхідним.

Таким чином, розглядаємо два сценарії реалізації програмного продукту.

1. $F_1B – F_2A – F_3A$.
2. $F_1B – F_2A – F_3B$.

Перейдемо до детальнішого обґрунтування нашого вибору.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні технологічні та програмні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри.

1. X1 – швидкодія мови програмування.

2. X2 – об’єм пам’яті, необхідний для керування даними.

3. X3 – час надання аналітичних результатів.

4. X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту.

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	70	110	140
Об’єм пам’яті, необхідний для керування даними	X2	мб	64	32	16
Час надання аналітичних результатів	X3	мс	1000	400	150
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	2500	1800

За даними таблиці 4.2 будуються графічні характеристики параметрів (рисунок 4.2 – рисунок 4.5).

Рисунок 4.2 – X1, швидкодія мови програмування.

Рисунок 4.3 – X2, об’єм пам’яті, необхідний для керування даними.

Рисунок 4.4 – X3, час надання аналітичних результатів.

Рисунок 4.5 – X4, потенційний об’єм програмного коду.

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі реалізації програмного продукту – розробка цифрового помічника, який дає зручний інтерфейс для швидкого зберігання та редагування нотаток, надання можливості автоматично визначати категорію і функціонал представлення аналітики ефективності користувача.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає 4 етапи.

1. Визначення рівня значимості параметра шляхом присвоєння різних рангів.
2. Перевірку придатності експертних оцінок для подальшого використання.
3. Визначення оцінки попарного пріоритету параметрів.
4. Обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів.

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	4	4	3	4	3	3	4	25	7.5	56.25
X2	Об’єм пам’яті, необхідний для керування даними	мб	2	2	2	1	1	1	2	11	-6.5	42.25
X3	Час надання аналітичних результатів	мс	3	3	4	3	4	4	3	24	6.5	42.25
X4	Потенційний об’єм програмного коду	К-ть рядків коду	1	1	1	2	2	2	1	10	-7.5	56.25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

1) сума рангів кожного з параметрів і загальна сума рангів за формулою 4.1.

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

де N – число експертів;

n – кількість параметрів;

2) середня сума рангів за формулою 4.2.

$$T = \frac{1}{n} R_{ij} = 17.5; \tag{4.2}$$

3) відхилення суми рангів кожного параметра від середньої суми рангів за формулою 4.3.

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

Сума відхилень по всім параметрам повинна дорівнювати 0;

4) загальна сума квадратів відхилення за формулою 4.4.

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості за формулою 4.5.

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0.804 > W_k = 0.67. \tag{4.5}$$

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1.5
X1 і X3	>	>	<	>	<	<	>	>	1.5
X1 і X4	>	>	>	>	>	>	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
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 за формулою 4.6.

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

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

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

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

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

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів.

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	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	1.5	1.5	1.5	5.5	0.34	21.25	0.36	77.875	0.36
X2	0.5	1	0.5	1.5	3.5	0.22	12.25	0.21	44.875	0.21
X3	0.5	1.5	1	1.5	4.5	0.28	16.25	0.27	59.125	0.27
X4	0.5	0.5	0.5	1	2.5	0.16	9.25	0.16	34.125	0.16
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Оцінка якості виконання кожного варіанту реалізації функцій програмного забезпечення досліджується через абсолютні значення відповідних параметрів, таких як об'єм пам'яті, необхідний для керування даними (X2), час надання аналітичних результатів (X3) та потенційний об'єм програмного коду (X4), що відповідають вимогам технічних умов функціонування даного ПЗ. Відмітимо, що параметр швидкодії мови програмування (X1) має показувати задовільний рівень продуктивності.

Для кожного варіанта реалізації програмного забезпечення нашої програми обчислюється показник якості, який розраховується за формулою 4.11.

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.11}$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

Використаємо дані з таблиці 4.6 і формулу 4.12.

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \tag{4.12}$$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП.

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	130	24	0.36	8.64
F2	А	X2	24	20	0.21	4.2
F3	А	X3	400	14	0.27	3.78
	Б	X4	1900	6	0.16	0.96

Тепер визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 8.64 + 4.2 + 3.78 = 16.62,$$

$$K_{K2} = 8.64 + 4.2 + 0.96 = 13.8.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 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.

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

Загальна трудомісткість обчислюється як:

$$\mathrm{T_O} = \mathrm{T_P} \cdot \mathrm{K_П} \cdot \mathrm{K_{СК}} \cdot \mathrm{K_М} \cdot \mathrm{K_{СТ}} \cdot \mathrm{K_{СТ.М}}, \qquad (4.13)$$

де $\mathrm{T_P}$ – трудомісткість розробки ПП;

$\mathrm{K_П}$ – поправочний коефіцієнт;

$\mathrm{K_{СК}}$ – коефіцієнт на складність вхідної інформації;

$\mathrm{K_М}$ – коефіцієнт рівня мови програмування;

$\mathrm{K_{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$\mathrm{K_{СТ.М}}$ – 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: $Т_Р$ = 50 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.2. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює: $Т_1$ = 50 · 1.2 · 0.8 = 48 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 40 людино-днів, $К_П$ = 1.4, $К_{СК}$ = 1, $К_{СТ}$ = 0.9:

$$Т_2 = 40 \cdot 1.4 \cdot 0.9 = 50.4 \text{ людино-днів.}$$

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

$$Т_I = (48 + 50.4 + 16.62) \cdot 8 = 920.16 \text{ людино-годин.}$$

$$Т_{II} = (48 + 50.4 + 13.8) \cdot 8 = 897.6 \text{ людино-годин.}$$

У розробці бере участь один програміст з окладом 25000 грн та один експерт з аналітики з окладом 24000. Визначимо середню зарплату за годину за формулою 4.14.

$$С_ч = \frac{М}{T_m \cdot t} \text{грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

$Т_m$ – кількість робочих днів тиждень;

t – кількість робочих годин в день.

Маємо для нашого варіанту:

$$С_ч = \frac{24000 + 25000}{2 \cdot 20 \cdot 8} = 153.125 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою 4.15.

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \qquad (4.15)$$

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 153.125 \cdot 920.2 \cdot 1.1 = 154989.5$ грн.

II. $С_{ЗП} = 153.125 \cdot 897.6 \cdot 1.1 = 151189.5$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 154989.5 \cdot 0.22 = 34097.68$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 151189.5 \cdot 0.22 = 33261.69$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0.3 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 25000 \cdot 0.2 = 60000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1+ К_З) = 60000 \cdot (1 + 0.2) = 72000 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 72000 \cdot 0.22 = 15840 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 26000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.1 \cdot 0.25 \cdot 26000 = 7150 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.1 \cdot 26000 \cdot 0.05 = 1430 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 11 – 10) \cdot 8 \cdot 0.75 = 1440$ години,

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1440 \cdot 0.25 \cdot 0.9 \cdot 9.43 = 3055.32 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 26000 \cdot 0.67 = 17420 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть обчислюватись за формулою 4.16.

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.16)$$

$$С_{ЕКС} = 72000 + 15840 + 7150 + 1430 + 3055.32 + 17420 = 116895.32 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 116895.32 / 1440 = 81.18 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу обчислюються за формулою 4.17.

$$С_М = С_{М-Г} \cdot Т, \quad (4.17)$$

І в залежності від обраного варіанта реалізації, складають:

I. $С_М = 81.18 \cdot 920.2 = 74698.588$ грн.

II. $С_М = 81.18 \cdot 897.6 = 72867.168$ грн.

Накладні витрати складають 67% від заробітної плати і обчислюються за формулою 4.18.

$$С_Н = С_{ЗП} \cdot 0.67, \quad (4.18)$$

Звідки маємо:

I. $C_Н = 154989.5 \cdot 0.67 = 103842.965$ грн.

II. $C_Н = 151189.5 \cdot 0.67 = 101296.965$ грн.

Вартість розробки ПП обчислюється за формулою 4.19.

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.19)$$

Отже, для нашого варіанту становить:

I. $C_{ПП} = 154989.5 + 34097.68 + 74698.588 + 103842.965 = 367628.733$ грн.

II. $C_{ПП} = 151189.5 + 33261.69 + 72867.168 + 101296.965 = 358615.323$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою 4.20.

$$К_{ТЕР j} = К_{Кj} / C_{Фj}, \quad (4.20)$$

Що для наших варіантів становитиме:

$$К_{ТЕР1} = \frac{16.62}{367628.733} = 4.5208 \cdot 10^{-5},$$

$$К_{ТЕР2} = \frac{13.8}{358615.323} = 3.8481 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 4.5208 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, які залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту.

Цей варіант реалізації програмного продукту має такі параметри:

1) вибір мови сервера – C++;
2) вибір мови фронту – Python;
3) вибір основної математичної бібліотеки – SciPy.

Даний варіант виконання програмного комплексу дає користувачу зручну реалізацію програми, зрозумілий інтерфейс, швидку роботу сервера і ефективний та інформативний аналітичний модуль.

4.8 Висновки до четвертого розділу

Було досліджено ключові фінансові аспекти розробки застосунку для підвищення особистої ефективності, зокрема його функціональні можливості та економічні показники.

Метою аналізу було визначити, які функції є найважливішими для користувача, оцінити їх вплив на загальну ефективність системи та зрозуміти, які ресурси необхідні для їх реалізації.

Це дослідження дозволило виявити оптимальний баланс між корисністю програмного рішення та витратами на його розробку. На основі отриманих даних було обрано найефективніший підхід до впровадження системи, який забезпечує високу продуктивність при мінімальних витратах.

ВИСНОВКИ

У ході виконання роботи було створено програмний застосунок для роботи з текстовими нотатками, який дозволяє додавати, редагувати та видаляти записи, автоматично визначати їхню категорію та аналізувати виконання за часом. Реалізований функціонал включає збереження нотаток у базі даних, визначення категорій нових нотаток та формування статистики за категоріями.

Для автоматичної категоризації було впроваджено кілька методів обчислення подібності між текстами. На основі вже існуючих записів нові нотатки отримують ймовірні категорії, що дозволяє автоматично віднести нову нотатку до відповідної групи. Також реалізовано обчислення середнього часу виконання завдань у межах кожної категорії та підрахунок випадків дотримання і порушення дедлайнів.

Застосунок має клієнт-серверну архітектуру: серверну частину реалізовано мовою C++, клієнтську — мовою Python. Обмін даними здійснюється через API на основі міжпроцесної комунікації із використанням сокетів на локальній машині. Створено графічний інтерфейс, що забезпечує доступ до функціоналу через інтерактивні елементи керування у вікні програми.

У процесі тестування було проведено верифікацію й валідацію програмного продукту відповідно до вимог, сформульованих у роботі. Техніко-економічний аналіз обґрунтував використання вибраних підходів та інструментів.

Таким чином, було реалізовано програмний засіб, який забезпечує роботу з нотатками через графічний інтерфейс користувача, а також надає засоби аналітики категорій та автоматичної класифікації записів за змістом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The History of NoteStack. *NoteStack Blog.* URL: https://notestack.app/blog/history.html (last accessed: 23.04.2025).
2. A Brief History of Note-Taking Apps: 2003-2022). *Photes.* URL: https://photes.io/notes/53c87b44-0fcf-4a24-a97a-1cc34ad7c14a/a-brief-history-of-note-taking-apps-2003-2022 (last accessed: 23.04.2025).
3. Artificial Intelligence + AI Revolutionizes the Classic Notepad Experience. *Medium.* URL: https://aiplusinfo.medium.com/ai-revolutionizes-the-classic-notepad-experience-fe0816188e89 (last accessed: 24.04.2025).
4. Guadagnolo L. Mastering Text Similarity: Combining Embedding Techniques and Distance Metrics. *Medium.* URL: https://medium.com/eni-digitalks/mastering-text-similarity-combining-embedding-techniques-and-distance-metrics-98d3bb80b1b6 (last accessed: 24.04.2025).
5. Hyndman R. J., Athanasopoulos G. Forecasting: principles and practice 2nd Edition. Melbourne, Australia: OTexts, 2018. 382 p.
6. Sarkar D. Text Analytics with Python: A Practitioner's Guide to Natural Language Processing. Bangalore, India: Apress Media, LLC, 2016. 397 p.
7. Алгоритм TF-IDF і визначення релевантності тексту. URL: https://expans.ua/blog/algorytm-tf-idf-i-vyznachennya-relevantnosti-tekstu/ (дата звернення: 27.04.2025).
8. Ultimate Guide To Text Similarity With Python. *Newscatcher.* URL: https://www.newscatcherapi.com/blog/ultimate-guide-to-text-similarity-with-python (last accessed: 25.04.2025).
9. What is latent semantic analysis?. *IBM.* URL: *https://www.ibm.com/think/topics/latent-semantic-analysis* (last accessed: 30.04.2025).
10. Kreibich J. A. Using SQLite: A Reliable Database Solution. Sebastopol, CA: O'Reilly Media, 2010. 526 p.

11. Bulka D., Mayhew D. Efficient C++: Performance Programming Techniques. Boston, MA: Addison-Wesley, 1999. 336 p.
12. C++ reference: A complete online reference for the C and C++ languages and standard libraries. *C++ reference.* URL: https://en.cppreference.com/w/ (last accessed: 04.05.2025).
13. Barbara J. Practical Backend Programming: Databases, High-Performance,and Best Practices. India: GitforGits. 2023. 275 p.
14. Python GUI Programming With Tkinter. *Real Python.* URL: https://realpython.com/python-gui-tkinter/ (last accessed: 28.04.2025).
15. Moore A. D. Python GUI Programming with Tkinter: Develop Responsive and Powerful GUI Applications. Birmingham, UK: Packt Publishing, 2018.
16. JSON for Modern C++. *GitHub.* URL: https://github.com/nlohmann/json (last accessed: 02.05.2025).
17. Libsqlite3 C++ library. *GitHub.* URL: https://github.com/LuaDist/libsqlite3 (last accessed: 01.05.2025).
18. Tkinter: Python interface to Tcl/Tk. *Python Documentation.* URL: https://docs.python.org/uk/3.13/library/tkinter.html (last accessed: 28.04.2025).
19. Matplotlib: Visualization with Python. бібліотека для візуалізації даних у Python. *Matplotlib.* URL: https://matplotlib.org (last accessed: 29.04.2025).
20. SciPy: Fundamental algorithms for scientific computing in Python. *SciPy.* URL: https://scipy.org (last accessed: 30.04.2025).
21. Scikit-learn: Machine Learning in Python. *Scikit-learn.* URL: https://scikit-learn.org/stable/ (last accessed: 30.04.2025).
22. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley, 1994. 417 p.
23. LogoAI: Logo & Brand Identity Design. *LogoAI.* URL: https://www.logoai.com/logo-maker (last accessed: 11.04.2025).

24. Scott C. Professional CMake: A Practical Guide. *Crascit.* URL: https://crascit.com/professional-cmake/ (last accessed: 08.05.2025).

ДОДАТОК А ЛІСТИНГ КОДУ

socket_handler.cpp:

```
#include "socket_handler.hpp"

SocketServer::SocketServer(int port,
DatabaseManager& db) : port(port),
db_manager(db), running(false)
{
    setup_socket();
}

SocketServer::~SocketServer()
{
    stop();
}

void SocketServer::setup_socket()
{
    if ((server_fd = socket(AF_INET,
SOCK_STREAM, 0)) == 0)
    {
        throw std::runtime_error("Socket
creation failed");
    }

    int opt = 1;
    if (setsockopt(server_fd, SOL_SOCKET,
SO_REUSEADDR, &opt, sizeof(opt)))
    {
        throw std::runtime_error("setsockopt
failed");
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(port);

    if (bind(server_fd, (struct sockaddr
*)&address, sizeof(address)) < 0)
    {
        throw std::runtime error("Bind
failed");
    }

    if (listen(server_fd, 5) < 0)
    {
        throw std::runtime_error("Listen
failed");
    }

    std::cout << "Socket server set up on port
" << port << std::endl;
}

void SocketServer::start()
{
    running = true;
    run_server_loop();
}

void SocketServer::stop()
{
    running = false;

    for (int client_socket : client_sockets)
    {
        close(client_socket);
    }
    client_sockets.clear();

    if (server_fd >= 0)
    {
        close(server_fd);
        server_fd = -1;
    }
}

void SocketServer::run_server_loop()
{
    int addrlen = sizeof(address);
    fd_set read_fds;
    int max_fd;

    std::cout << "Waiting for connections..."
<< std::endl;

    while (running)
    {
        FD_ZERO(&read_fds);

        FD_SET(server_fd, &read_fds);
        max_fd = server_fd;

        for (int client_socket :
client_sockets)
        {
            FD_SET(client_socket, &read_fds);
            if (client_socket > max_fd)
            {
                max_fd = client_socket;
            }
        }

        struct timeval timeout;
        timeout.tv_sec = 1;
        timeout.tv_usec = 0;

        int activity = select(max_fd + 1,
&read_fds, NULL, NULL, &timeout);

        if (activity < 0 && errno != EINTR)
        {
            std::cerr << "Select error" <<
std::endl;
            continue;
        }

        if (activity == 0)
        {
            continue;
        }

        if (FD_ISSET(server_fd, &read_fds))
        {
            int new_socket = accept(server_fd,
(struct sockaddr *)&address,
(socklen_t*)&addrlen);
            if (new_socket < 0)
            {
                std::cerr << "Accept failed"
<< std::endl;
                continue;
            }

            std::cout << "New client
connected" << std::endl;

client_sockets.push_back(new_socket);
        }
```

```
        for (auto it = client_sockets.begin();
it != client_sockets.end();)
        {
            int client_socket = *it;

            if (FD_ISSET(client_socket,
&read_fds))
            {
                if
(!process_client_data(client_socket))
                {
                    close(client_socket);
                    std::cout << "Client
disconnected" << std::endl;
                    it =
client_sockets.erase(it);
                    continue;
                }
            }
            ++it;
        }
    }
}

bool SocketServer::process_client_data(int
client_socket)
{
    char buffer[BUFFER_SIZE] = {0};
    static std::map<int, std::string>
message_buffers;

    int bytes_read = recv(client_socket,
buffer, BUFFER_SIZE - 1, 0);

    if (bytes_read <= 0)
    {
        return false;
    }

    buffer[bytes_read] = '\0';
    message_buffers[client_socket] += buffer;

    size_t pos;
    while ((pos =
message_buffers[client_socket].find('\n')) !=
std::string::npos)
    {
        std::string message =
message_buffers[client_socket].substr(0, pos);
        message_buffers[client_socket] =
message_buffers[client_socket].substr(pos +
1);

        try
        {
            handle_message(client_socket,
message);
        }
        catch (const std::exception& e)
        {
            std::cerr << "Error handling
message: " << e.what() << std::endl;

            json error_response =
            {
                {"status", "error"},
                {"message", e.what()}
            };
            send_response(client_socket,
error_response);
        }

        memset(buffer, 0, BUFFER_SIZE);
    }

    return true;
}

void SocketServer::handle_message(int
client_socket, const std::string& message)
{
    json request = json::parse(message);
    std::string operation =
request["operation"];

    std::cout << "Received operation: " <<
operation << std::endl;

    if (operation == "save")
    {
        Note note =
Note::from_json(request["note"]);
        int note_id =
db_manager.save_note(note);

        json response =
        {
            {"status", "success"},
            {"operation", "save"},
            {"note_id", note_id}
        };
        send_response(client_socket,
response);
    }
    else if (operation == "delete")
     {
        int note_id = request["note_id"];
        bool success =
db_manager.delete_note(note_id);

        json response =
        {
            {"status", success ? "success" :
"error"},
            {"operation", "delete"},
            {"note_id", note_id}
        };
        send_response(client_socket,
response);
    }
    else if (operation == "get")
    {
        int note_id = request["note_id"];

        try
        {
            Note note =
db_manager.get_note(note_id);

            json response =
            {
                {"status", "success"},
                {"operation", "get"},
                {"note", note.to_json()}
            };
            send_response(client_socket,
response);
        }
        catch (const std::exception& e)
        {
            json response =
            {
                {"status", "error"},
                {"operation", "get"},
                {"message", e.what()}
            };
            send_response(client_socket,
response);
        }
    }
    else if (operation == "list")
    {
        std::vector<Note> notes =
db_manager.get_all_notes();

        json notes_json = json::array();
        for (const auto& note : notes)
        {

notes_json.push_back(note.to_json());
```

```
        }

        json response =
        {
            {"status", "success"},
            {"operation", "list"},
            {"notes", notes_json}
        };
        send_response(client_socket,
response);
    }
    else
    {
        throw std::runtime_error("Unknown
operation: " + operation);
    }
}

void SocketServer::send_response(int
client_socket, const json& response)
{
    std::string response_str = response.dump()
+ "\n";
    send(client_socket, response_str.c_str(),
response_str.length(), 0);
}
```

socket_handler.hpp:

```
#ifndef SOCKET_HANDLER_HPP
#define SOCKET HANDLER HPP

#include
"../DataBaseManager/data_base_manager.hpp"

class SocketServer
{
    private:
        int port;
        int server_fd = -1;
        struct sockaddr_in address;
        bool running;
        std::vector<int> client_sockets;
        DatabaseManager& db_manager;

        void setup_socket();
        void run_server_loop();
        bool process_client_data(int
client_socket);
        void handle_message(int client_socket,
const std::string& message);
        void send_response(int client_socket,
const json& response);

    public:
        SocketServer(int port,
DatabaseManager& db);
        ~SocketServer();

        void start();
        void stop();
};

#endif
```

data_base_manager.cpp:

```
#include "data_base_manager.hpp"

// Note structure that mirrors the Python Note
class
json Note::to_json() const
{
    json j;
    j["id"] = id;
    j["title"] = title;
    j["content"] = content;
    j["date"] = date;
    j["category"] = category;
    j["status"] = status;
    j["deadline"] = deadline;
    j["completion_date"] = completion_date;
    return j;
}

// Create from JSON
Note Note::from_json(const json& j)
{
    Note note;

    if (j.contains("id") &&
!j["id"].is_null())
    {
        note.id = j["id"].get<int>();
    }
    else
    {
        note.id = -1;
    }

    note.title =
j["title"].get<std::string>();
    note.content =
j["content"].get<std::string>();
    note.date = j["date"].get<std::string>();
    note.category =
j["category"].get<std::string>();
    note.status =
j["status"].get<std::string>();
    note.deadline =
j["deadline"].get<std::string>();
    note.completion_date =
j["completion_date"].get<std::string>();
    return note;
}

DatabaseManager::DatabaseManager(const
std::string& db_path) : database_path(db_path)
{
    init_database();
}

void DatabaseManager::init_database()
{
    try
    {
        int rc =
sqlite3_open(database_path.c_str(), &db);
        if (rc != SQLITE_OK)
        {
            std::string error_msg = "Cannot
open database: " +
std::string(sqlite3_errmsg(db));
            sqlite3_close(db);
            throw
std::runtime_error(error_msg);
        }

        const char* create_table_sql =
            "CREATE TABLE IF NOT EXISTS notes
("
            "id INTEGER PRIMARY KEY
AUTOINCREMENT, "
            "title TEXT NOT NULL, "
            "content TEXT NOT NULL, "
            "date TEXT NOT NULL, "
            "category TEXT, "
            "status TEXT DEFAULT 'normal', "
            "deadline TEXT, "
            "completion_date TEXT"
            ");";

        char* error_message = nullptr;
        rc = sqlite3_exec(db,
create_table_sql, nullptr, nullptr,
&error_message);
```

```
        if (rc != SQLITE_OK)
        {
            std::string error_msg = "SQL
error: " + std::string(error_message);
            sqlite3_free(error_message);
            throw
std::runtime_error(error_msg);
        }

        sqlite3_close(db);
        std::cout << "Database initialized
successfully" << std::endl;
    }
    catch (const std::exception& e)
    {
        std::cerr << "Database initialization
error: " << e.what() << std::endl;
        throw;
    }
}

std::vector<Note>
DatabaseManager::get_all_notes()
{
    std::vector<Note> notes;

    try
    {
        sqlite3* db;
        if
(sqlite3_open(database_path.c_str(), &db) !=
SQLITE_OK)
        {
            throw std::runtime_error("Cannot
open database");
        }

        const char* query = "SELECT id, title,
content, date, category, status, deadline,
completion_date FROM notes ORDER BY date
DESC";
        sqlite3_stmt* stmt;

        if (sqlite3_prepare_v2(db, query, -1,
&stmt, nullptr) != SQLITE_OK)
        {
            std::string error_msg = "Failed to
prepare statement: " +
std::string(sqlite3_errmsg(db));
            sqlite3_close(db);
            throw
std::runtime_error(error_msg);
        }

        while (sqlite3_step(stmt) ==
SQLITE_ROW)
        {
            Note note;
            note.id = sqlite3_column_int(stmt,
0);
            note.title =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 1));
            note.content =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2));
            note.date = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 3));

            if (sqlite3_column_type(stmt, 4)
!= SQLITE_NULL)
            {
                note.category =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 4));
            }
            else
            {
                note.category = "";
            }

            if (sqlite3_column_type(stmt, 5)
!= SQLITE_NULL)
            {
                note.status =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 5));
            }
            else
            {
                note.status = "normal";
            }

            if (sqlite3_column_type(stmt, 6)
!= SQLITE_NULL)
            {
                note.deadline =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 6));
            }
            else
            {
                note.deadline = "";
            }

            if (sqlite3_column_type(stmt, 7)
!= SQLITE_NULL)
            {
                note.completion_date =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 7));
            }
            else
            {
                note.completion_date = "";
            }

            notes.push_back(note);
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
    }
    catch (const std::exception& e)
    {
        std::cerr << "Error retrieving notes:
" << e.what() << std::endl;
    }

    return notes;
}

Note DatabaseManager::get_note(int id)
{
    Note note;

    try
    {
        sqlite3* db;
        if
(sqlite3_open(database_path.c_str(), &db) !=
SQLITE_OK)
        {
            throw std::runtime error("Cannot
open database");
        }

        const char* query = "SELECT id, title,
content, date, category, status, deadline,
completion_date FROM notes WHERE id = ?";
        sqlite3_stmt* stmt;

        if (sqlite3_prepare_v2(db, query, -1,
&stmt, nullptr) != SQLITE_OK)
        {
            std::string error_msg = "Failed to
prepare statement: " +
std::string(sqlite3_errmsg(db));
            sqlite3_close(db);
            throw
```

```
std::runtime_error(error_msg);
        }

        sqlite3_bind_int(stmt, 1, id);

        if (sqlite3_step(stmt) == SQLITE_ROW)
        {
            note.id = sqlite3_column_int(stmt,
0);
            note.title =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 1));
            note.content =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2));
            note.date = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 3));

            if (sqlite3_column_type(stmt, 4)
!= SQLITE_NULL)
            {
                note.category =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 4));
            }
            else
            {
                note.category = "";
            }

            if (sqlite3_column_type(stmt, 5)
!= SQLITE_NULL)
            {
                note.status =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 5));
            }
            else
            {
                note.status = "normal";
            }

            if (sqlite3_column_type(stmt, 6)
!= SQLITE_NULL)
            {
                note.deadline =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 6));
            }
            else
            {
                note.deadline = "";
            }

            if (sqlite3_column_type(stmt, 7)
!= SQLITE_NULL)
            {
                note.completion_date =
reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 7));
            }
            else
            {
                note.completion_date = "";
            }
        }
        else
        {
            sqlite3_finalize(stmt);
            sqlite3_close(db);
            throw std::runtime_error("Note not
found");
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
    }
    catch (const std::exception& e)
    {
        std::cerr << "Error retrieving note: "
<< e.what() << std::endl;
        throw;
    }

    return note;
}

int DatabaseManager::save_note(const Note&
note)
{
    int note_id = note.id;

    try
    {
        sqlite3* db;
        if
(sqlite3_open(database_path.c_str(), &db) !=
SQLITE_OK)
        {
            throw std::runtime_error("Cannot
open database");
        }

        if (note_id <= 0)
        {
            const char* insert_sql = "INSERT
INTO notes (title, content, date, category,
status, deadline, completion_date) "
                                     "VALUES
(?, ?, ?, ?, ?, ?, ?)";
            sqlite3_stmt* stmt;

            if (sqlite3_prepare_v2(db,
insert_sql, -1, &stmt, nullptr) != SQLITE_OK)
            {
                std::string error_msg =
"Failed to prepare statement: " +
std::string(sqlite3_errmsg(db));
                sqlite3_close(db);
                throw
std::runtime_error(error_msg);
            }

            sqlite3_bind_text(stmt, 1,
note.title.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 2,
note.content.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 3,
note.date.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 4,
note.category.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 5,
note.status.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 6,
note.deadline.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 7,
note.completion_date.c_str(), -1,
SQLITE STATIC);

            if (sqlite3_step(stmt) !=
SQLITE_DONE)
            {
                std::string error_msg =
"Failed to insert note: " +
std::string(sqlite3_errmsg(db));
                sqlite3_finalize(stmt);
                sqlite3_close(db);
                throw
std::runtime_error(error_msg);
            }

            note_id =
sqlite3_last_insert_rowid(db);
            sqlite3_finalize(stmt);
        }
        else
        {
            const char* update_sql = "UPDATE
notes SET title=?, content=?, date=?,
category=?, status=?, deadline=?,
```

```
completion_date=? "
                                    "WHERE
id=?";
            sqlite3_stmt* stmt;

            if (sqlite3_prepare_v2(db,
update_sql, -1, &stmt, nullptr) != SQLITE_OK)
            {
                std::string error_msg =
"Failed to prepare statement: " +
std::string(sqlite3_errmsg(db));
                sqlite3_close(db);
                throw
std::runtime_error(error_msg);
            }

            sqlite3_bind_text(stmt, 1,
note.title.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 2,
note.content.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 3,
note.date.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 4,
note.category.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 5,
note.status.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 6,
note.deadline.c_str(), -1, SQLITE_STATIC);
            sqlite3_bind_text(stmt, 7,
note.completion_date.c_str(), -1,
SQLITE_STATIC);
            sqlite3_bind_int(stmt, 8,
note_id);

            if (sqlite3_step(stmt) !=
SQLITE_DONE)
            {
                std::string error_msg =
"Failed to update note: " +
std::string(sqlite3_errmsg(db));
                sqlite3_finalize(stmt);
                sqlite3_close(db);
                throw
std::runtime_error(error_msg);
            }

            sqlite3_finalize(stmt);
        }

        sqlite3_close(db);
    }
    catch (const std::exception& e)
    {
        std::cerr << "Error saving note: " <<
e.what() << std::endl;
        throw;
    }

    return note_id;
}

bool DatabaseManager::delete_note(int id)
{
    try
    {
        sqlite3* db;
        if
(sqlite3_open(database_path.c_str(), &db) !=
SQLITE_OK)
        {
            throw std::runtime_error("Cannot
open database");
        }

        const char* delete_sql = "DELETE FROM
notes WHERE id = ?";
        sqlite3_stmt* stmt;

        if (sqlite3_prepare_v2(db, delete_sql,
-1, &stmt, nullptr) != SQLITE_OK)
        {
            std::string error_msg = "Failed to
prepare statement: " +
std::string(sqlite3_errmsg(db));
            sqlite3_close(db);
            throw
std::runtime_error(error_msg);
        }

        sqlite3_bind_int(stmt, 1, id);

        if (sqlite3_step(stmt) != SQLITE_DONE)
        {
            std::string error_msg = "Failed to
delete note: " +
std::string(sqlite3_errmsg(db));
            sqlite3_finalize(stmt);
            sqlite3_close(db);
            throw
std::runtime_error(error_msg);
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
        return true;
    }
    catch (const std::exception& e)
    {
        std::cerr << "Error deleting note: "
<< e.what() << std::endl;
        return false;
    }
}
```

data_base_manager.hpp:

```
#ifndef DATA_BASE_MANAGER_HPP
#define DATA_BASE_MANAGER_HPP

#include <string>
#include <vector>
#include <iostream>
#include <algorithm>
#include <stdexcept>
#include <sqlite3.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <nlohmann/json.hpp>

using json = nlohmann::json;

#define BUFFER_SIZE 4096

// Note structure
struct Note
{
    int id = -1;
    std::string title;
    std::string content;
    std::string date;
    std::string category;
    std::string status = "normal";
    std::string deadline;
    std::string completion date;

    json to_json() const;
    static Note from_json(const json& j);
};

// Database manager for SQLite
class DatabaseManager
{
    private:
        std::string database_path;
        sqlite3* db = nullptr;

        void init_database();

    public:
```

```
        DatabaseManager(const std::string&
db_path);

        std::vector<Note> get_all_notes();
        Note get_note(int id);
        int save_note(const Note& note);
        bool delete_note(int id);
};

#endif
```

main.cpp:

```
#include
"ConnectionHandler/socket_handler.hpp"
#include <iostream>
#include <string>
#include <csignal>
#include <cstdlib>
#include <thread>

SocketServer* g_server = nullptr;

void signal_handler(int signal) {
    std::cout << "\nReceived signal " <<
signal << ". Shutting down server..." <<
std::endl;
    if (g_server) {
        g_server->stop();
    }
    exit(signal);
}

void print_usage(const char* program_name) {
    std::cout << "Usage: " << program_name <<
" [options]" << std::endl;
    std::cout << "Options:" << std::endl;
    std::cout << "  --port PORT       Specify
the port to listen on (default: 8080)" <<
std::endl;
    std::cout << "  --db PATH         Specify
the SQLite database path (default: notes.db)"
<< std::endl;
    std::cout << "  --help            Display
this help message" << std::endl;
}

int main(int argc, char* argv[]) {
    int port = 8080;
    std::string db_path = "note_db/notes.db";

    for (int i = 1; i < argc; i++) {
        std::string arg = argv[i];

        if (arg == "--help") {
            print_usage(argv[0]);
            return 0;
        } else if (arg == "--port" && i + 1 <
argc) {
            port = std::atoi(argv[++i]);
            if (port <= 0 || port > 65535) {
                std::cerr << "Invalid port
number. Must be between 1 and 65535." <<
std::endl;
                return 1;
            }
        } else if (arg == "--db" && i + 1 <
argc) {
            db_path = argv[++i];
        } else {
            std::cerr << "Unknown option: " <<
arg << std::endl;
            print_usage(argv[0]);
            return 1;
        }
    }

    try {
        std::cout << "Starting Notes
Server..." << std::endl;
        std::cout << "Database: " << db_path
<< std::endl;
        std::cout << "Port: " << port <<
std::endl;

        std::signal(SIGINT, signal_handler);
        std::signal(SIGTERM, signal_handler);

        DatabaseManager db_manager(db_path);

        SocketServer server(port, db_manager);
        g_server = &server;

        server.start();

        std::cout << "Server started. Press
Ctrl+C to stop." << std::endl;

        while (true) {

std::this_thread::sleep_for(std::chrono::secon
ds(1));
        }
    }
    catch (const std::exception& e) {
        std::cerr << "Fatal error: " <<
e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.10)
project(socket_handler)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(PkgConfig REQUIRED)

find_package(SQLite3 REQUIRED)
if(SQLite3_FOUND)

include_directories(${SQLite3_INCLUDE_DIRS})
else()
    pkg_check_modules(SQLite3 REQUIRED
sqlite3)
    if(SQLite3_FOUND)

include_directories(${SQLite3_INCLUDE_DIRS})

link_directories(${SQLite3_LIBRARY_DIRS})
    else()
        message(FATAL_ERROR "SQLite3
development libraries not found. Please
install them.")
    endif()
endif()

find_package(nlohmann_json REQUIRED)

add executable(socket handler main.cpp
ConnectionHandler/socket_handler.cpp
DataBaseManager/data_base_manager.cpp)
target_link_libraries(socket_handler
${SQLite3_LIBRARIES}
nlohmann_json::nlohmann_json)
```

note.py:

```
from datetime import datetime


class Note:
    def   init  (self, id=0, title="",
content="", date=None, category="",
status="normal", deadline=None,
```

```
                completion_date=None):
        self.id = id
        self.title = title
        self.content = content
        self.date = date or
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.category = category
        self.status = status
        self.deadline = deadline
        self.completion_date = completion_date

    def is_overdue(self):
        if self.status != "todo" or not
self.deadline:
            return False

        try:
            deadline_date =
datetime.strptime(self.deadline, "%Y-%m-%d
%H:%M:%S")
            current_date = datetime.now()
            return current_date >
deadline_date
        except ValueError:
            return False

    def mark_as_done(self):
        self.status = "done"
        self.completion_date =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    def completion_time(self):
        if self.status != "done" or not
self.completion_date or not self.date:
            return None

        try:
            start_date =
datetime.strptime(self.date, "%Y-%m-%d
%H:%M:%S")
            end_date =
datetime.strptime(self.completion_date, "%Y-
%m-%d %H:%M:%S")
            time_diff = end_date - start_date
            return time_diff.total_seconds() /
3600
        except ValueError:
            return None

    def deadline_adherence(self):
        if self.status != "done" or not
self.completion_date or not self.deadline:
            return None

        try:
            deadline_date =
datetime.strptime(self.deadline, "%Y-%m-%d
%H:%M:%S")
            completion_date =
datetime.strptime(self.completion_date, "%Y-
%m-%d %H:%M:%S")
            time_diff = deadline_date -
completion_date
            return time diff.total seconds() /
3600
        except ValueError:
            return None
```

note_analytics.py:

```
import math
import numpy as np
from collections import defaultdict, Counter
from datetime import datetime, timedelta
import pandas as pd
from scipy import stats


class NotesAnalytics:
    # Service for analyzing Note statistics
and patterns

    def   init  (self):
        self.notes = []
        self.significance_level = 0.05

    def set_notes(self, notes):
        self.notes = notes

    def get_category_stats(self):
        if not self.notes:
            return {}

        categories = defaultdict(list)

        for note in self.notes:
            if note.category:

categories[note.category].append(note)

        results = {}
        for category, cat_notes in
categories.items():
            status_counts =
Counter([note.status for note in cat_notes])

            todo_done_count =
status_counts.get("todo", 0) +
status_counts.get("done", 0)
            completion_rate =
status_counts.get("done", 0) / max(1,
todo_done_count)

            adherence_values =
[note.deadline_adherence() for note in
cat_notes
                                    if note.status
== "done" and note.deadline_adherence() is not
None]

            completion_times =
[note.completion_time() for note in cat_notes
                                    if note.status
== "done" and note.completion_time() is not
None]

            results[category] = {
                "count": len(cat_notes),
                "status_distribution":
dict(status_counts),
                "completion_rate":
completion_rate,
                "avg_completion_time":
np.mean(completion_times) if completion_times
else None,
                "median_completion_time":
np.median(completion_times) if
completion times else None,
                "std_completion_time":
np.std(completion_times) if
len(completion_times) > 1 else None,
                "avg_deadline_adherence":
np.mean(adherence_values) if adherence_values
else None,
                "median_deadline_adherence":
np.median(adherence_values) if
adherence_values else None,
                "std_deadline_adherence":
np.std(adherence_values) if
len(adherence_values) > 1 else None,
                "on_time_rate": sum(1 for v in
adherence_values if v >= 0) / max(1,
len(adherence_values))
                    if adherence_values else None
            }

        return results

    def get_completion_speed(self):
        if not self.notes:
```

```
            return {}

        completed_notes = [note for note in
self.notes if note.status == "done" and
note.completion_time() is not None]
        if not completed_notes:
            return {}

        word_counts = []
        completion_times = []
        categories = []
        time_of_day = []

        for note in completed_notes:
            word_count =
len(note.content.split())
            completion_time =
note.completion_time()

            if completion_time is not None and
completion_time > 0:
                word_counts.append(word_count)

completion_times.append(completion_time)

categories.append(note.category)

                try:
                    create_date =
datetime.strptime(note.date, "%Y-%m-%d
%H:%M:%S")
                    hour = create_date.hour
                    if 5 <= hour < 12:
                        period = "morning"
                    elif 12 <= hour < 17:
                        period = "afternoon"
                    elif 17 <= hour < 22:
                        period = "evening"
                    else:
                        period = "night"
                    time_of_day.append(period)
                except ValueError:

time_of_day.append("unknown")

        if len(word_counts) >= 2:
            correlation, p_value =
stats.pearsonr(word_counts, completion_times)

            slope, intercept, r_value,
p_value, std_err =
stats.linregress(word_counts,
completion_times)

            regression = {
                "slope": slope,
                "intercept": intercept,
                "r squared": r value ** 2,
                "p_value": p_value,
                "std_err": std_err
            }

            def predict_time(word_count):
                return slope * word count +
intercept

            if categories:
                df = pd.DataFrame({
                    'category': categories,
                    'word_count': word_counts,
                    'completion_time':
completion_times,
                    'hours_per_word': [time /
max(1, count) for time, count in
zip(completion_times, word_counts)]
                })

                category_speeds =
df.groupby('category')['hours_per_word'].mean(
).sort_values()
                category_efficiency = {
                    cat: speed for cat, speed
in category speeds.items() if not
pd.isna(speed)
                }

                if time_of_day:
                    time df = pd.DataFrame({
                        'time_of_day':
time_of_day,
                        'hours_per_word':
[time / max(1, count) for time, count in
zip(completion_times, word_counts)]
                    })

                    time_efficiency =
time_df.groupby('time_of_day')['hours_per_word
'].mean().to_dict()

                    time_groups = []
                    for period in ['morning',
'afternoon', 'evening', 'night']:
                        group =
time_df[time_df['time_of_day'] ==
period]['hours_per_word'].values
                        if len(group) > 0:

time_groups.append(group)

                    anova_result = None
                    if len(time_groups) >= 2
and all(len(g) > 0 for g in time_groups):
                        try:
                            f_stat, p_value =
stats.f_oneway(*time_groups)
                            anova_result = {
                                'f_statistic':
f_stat,
                                'p_value':
p_value,
                                'significant':
p_value < self.significance_level
                            }
                        except:
                            anova_result =
None
                else:
                    time_efficiency = {}
                    anova_result = None

                df['predicted_time'] =
df['word_count'].apply(predict_time)
                df['time_difference'] =
df['completion_time'] - df['predicted_time']

                q1 =
df['time_difference'].quantile(0.25)
                q3 =
df['time_difference'].quantile(0.75)
                iqr = q3 - q1
                outlier_threshold = q3 + 1.5 *
iqr

                outliers =
df[df['time_difference'] > outlier_threshold]
                outlier_count = len(outliers)

                return {
                    "correlation":
correlation,
                    "regression": regression,
                    "category_efficiency":
category_efficiency,
                    "time_of_day_efficiency":
time_efficiency,
                    "time_of_day_anova":
anova_result,
                    "outliers": outlier_count,
                    "outlier_rate":
outlier_count / len(word_counts) if
```

```
word_counts else 0
                }

        return {}

    def get_deadline_compliance(self):
        if not self.notes:
            return {}

        deadline_notes = [note for note in
self.notes if note.deadline is not None]
        if not deadline_notes:
            return {}

        total_with_deadline =
len(deadline_notes)
        completed_notes = [note for note in
deadline_notes if note.status == "done"]
        completed_count = len(completed_notes)

        completion_rate = completed_count /
total_with_deadline if total_with_deadline > 0
else 0

        completed_on_time = [note for note in
completed_notes if

note.deadline_adherence() is not None and
note.deadline_adherence() >= 0]
        on_time_rate = len(completed_on_time)
/ completed_count if completed_count > 0 else
0

        adherence_values =
[note.deadline_adherence() for note in
completed_notes if

note.deadline_adherence() is not None]
        avg_adherence =
np.mean(adherence_values) if adherence_values
else 0

        category_adherence = defaultdict(list)
        for note in completed_notes:
            if note.category and
note.deadline_adherence() is not None:

category_adherence[note.category].append(note.
deadline_adherence())

        category_stats = {}
        for category, values in
category_adherence.items():
            if values:
                category_stats[category] = {
                    "avg_adherence":
np.mean(values),
                    "median adherence":
np.median(values),
                    "std_adherence":
np.std(values) if len(values) > 1 else 0,
                    "on_time_rate": sum(1 for
v in values if v >= 0) / len(values)
                }

        buffer_hours = []
        completed = []

        for note in deadline_notes:
            try:
                create_date =
datetime.strptime(note.date, "%Y-%m-%d
%H:%M:%S")
                deadline_date =
datetime.strptime(note.deadline, "%Y-%m-%d
%H:%M:%S")
                buffer = (deadline_date -
create_date).total_seconds() / 3600

                if buffer > 0:
buffer_hours.append(buffer)
                    completed.append(1 if
note.status == "done" else 0)
            except ValueError:
                continue

        buffer correlation = None
        if len(buffer_hours) >= 5:
            buffer_correlation, p_value =
stats.pointbiserialr(completed, buffer_hours)

            try:
                buffer_array =
np.array(buffer_hours).reshape(-1, 1)
                from sklearn.linear_model
import LogisticRegression
                model = LogisticRegression()
                model.fit(buffer_array,
completed)

                buffer_logit = {
                    "coefficient":
model.coef_[0][0],
                    "intercept":
model.intercept_[0],
                    "score":
model.score(buffer_array, completed),
                    "significant": p_value <
self.significance_level
                }
            except:
                buffer_logit = None
        else:
            buffer_logit = None

        return {
            "completion_rate":
completion_rate,
            "on_time_rate": on_time_rate,
            "avg_hours_to_deadline":
avg_adherence,
            "category_stats": category_stats,
            "buffer_correlation":
buffer_correlation,
            "buffer_logit": buffer_logit
        }

    def get_content_analysis(self):
        if not self.notes:
            return {}

        word_counts = []
        categories = []

        for note in self.notes:
            count = len(note.content.split())
            word counts.append(count)
            categories.append(note.category)

        avg_length = np.mean(word_counts) if
word_counts else 0
        median_length = np.median(word_counts)
if word counts else 0
        std_length = np.std(word_counts) if
len(word_counts) > 1 else 0

        category_lengths = defaultdict(list)
        for cat, count in zip(categories,
word_counts):
            if cat:

category_lengths[cat].append(count)

        cat_stats = {}
        for cat, lengths in
category_lengths.items():
            cat_stats[cat] = {
                "avg_length":
np.mean(lengths),
```

```
                "median_length":
np.median(lengths),
                "std length": np.std(lengths)
if len(lengths) > 1 else 0
            }

        length_stats = {
            "avg length": avg length,
            "median_length": median_length,
            "std_length": std_length,
            "by_category": cat_stats
        }

        text_vs_time = {}
        word_counts_completed = []
        completion_times = []

        for note in self.notes:
            if note.status == "done" and
note.completion_time() is not None:
                count =
len(note.content.split())

word_counts_completed.append(count)

completion_times.append(note.completion_time()
)

        if len(word_counts_completed) >= 2:
            correlation, p_value =
stats.pearsonr(word_counts_completed,
completion_times)

            slope, intercept, r_value,
p_value, std_err =
stats.linregress(word_counts_completed,
completion_times)

            text_vs_time = {
                "correlation": correlation,
                "p_value": p_value,
                "significant": p_value <
self.significance_level,
                "regression": {
                    "slope": slope,
                    "intercept": intercept,
                    "r_squared": r_value ** 2
                }
            }

        stopwords = {"the", "and", "a", "to",
"of", "in", "i", "is", "that", "it", "for",
"you", "this", "with", "on",
                     "be", "are", "as", "at",
"have", "from", "or", "not", "by", "but"}

        category_words = defaultdict(Counter)

        for note in self.notes:
            if note.category:
                words =
note.content.lower().split()
                clean_words =
[word.strip(".,!?:;\"'()[]{}") for word in
words]
                clean_words = [word for word
in clean_words if word and word not in
stopwords and len(word) > 2]


category_words[note.category].update(clean_wor
ds)

        word_freq = {}
        for category, counter in
category_words.items():
            top_words =
counter.most_common(10)

            tf = {word: count for word, count
in top_words}

            word categories = defaultdict(int)
            for cat, words in
category_words.items():
                for word in tf:
                    if word in words:
                        word categories[word]
+= 1

            total_categories =
len(category_words)
            tfidf = {}
            for word, count in tf.items():
                df = word_categories[word]
                idf =
math.log(total_categories / (df + 1))
                tfidf[word] = count * idf

            distinctive_words =
sorted(tfidf.items(), key=lambda x: x[1],
reverse=True)[:5]

            word_freq[category] = {
                "top_words": top_words,
                "distinctive_words":
distinctive_words
            }

        return {
            "length_stats": length_stats,
            "text_vs_time": text_vs_time,
            "word_freq": word_freq
        }

    def get_summary_statistics(self):
        if not self.notes:
            return {}

        status_counts = Counter([note.status
for note in self.notes])

        category_counts =
Counter([note.category for note in self.notes
if note.category])

        todo_done_count =
status_counts.get("todo", 0) +
status_counts.get("done", 0)
        completion_rate =
status_counts.get("done", 0) / max(1,
todo_done_count)

        completion_times =
[note.completion_time() for note in self.notes
                            if note.status ==
"done" and note.completion_time() is not None]
        avg completion time =
np.mean(completion_times) if completion_times
else None

        adherence_values =
[note.deadline_adherence() for note in
self.notes
                            if note.status ==
"done" and note.deadline_adherence() is not
None]
        on_time_rate = sum(1 for v in
adherence_values if v >= 0) / max(1,

len(adherence_values)) if adherence_values
else None

        return {
            "total_notes": len(self.notes),
            "status_counts":
dict(status_counts),
            "category_counts":
dict(category_counts),
            "completion_rate":
```

```
completion_rate,
            "avg_completion_time":
avg completion time,
            "on_time_rate": on_time_rate
        }

    def get_actionable_insights(self):
        insights = []

        category_stats =
self.get_category_stats()
        deadline_compliance =
self.get_deadline_compliance()
        completion_speed =
self.get_completion_speed()

        # Insight 1: Identify most efficient
category
        if completion_speed and
"category_efficiency" in completion_speed:
            category_efficiency =
completion_speed["category_efficiency"]
            if category_efficiency:
                most_efficient =
min(category_efficiency.items(), key=lambda x:
x[1])
                least_efficient =
max(category_efficiency.items(), key=lambda x:
x[1])

                insights.append({
                    "type": "efficiency",
                    "title": "Most Efficient
Category",
                    "description":
f"'{most_efficient[0]}' is your most efficient
category with an average of
{most_efficient[1]:.5f} hours per word.",
                    "comparison": f"This is
{least_efficient[1] / most_efficient[1]:.1f}x
faster than your least efficient category
'{least_efficient[0]}'."
                })

        # Insight 2: Identify best time of day
for productivity
        if (completion_speed and
"time_of_day_efficiency" in completion_speed
and

completion_speed["time_of_day_efficiency"]):

            time_efficiency =
completion_speed["time_of_day_efficiency"]
            best_time =
min(time_efficiency.items(), key=lambda x:
x[1])
            worst time =
max(time_efficiency.items(), key=lambda x:
x[1])

            insights.append({
                "type": "time_of_day",
                "title": "Best Time of Day",
                "description": f"You're most
productive during the {best_time[0]} with
{best_time[1]:.5f} hours per word on
average.",
                "comparison": f"This is
{worst_time[1] / best_time[1]:.1f}x faster
than your least productive time
({worst_time[0]})."
            })

            if
(completion_speed["time_of_day_anova"] and

completion_speed["time_of_day_anova"]["signifi
cant"]):
                insights[-1]["description"] +=
" This difference is statistically
significant."

        # Insight 3: Identify best performing
category for deadline compliance
        if deadline_compliance and
"category_stats" in deadline_compliance:
            category stats =
deadline_compliance["category_stats"]
            if category_stats:
                best_category =
max(category_stats.items(), key=lambda x:
x[1]["on_time_rate"])

                insights.append({
                    "type": "deadline",
                    "title": "Most Reliable
Category",
                    "description": f"Your
'{best_category[0]}' tasks have the highest
on-time completion rate at
{best_category[1]['on_time_rate'] *
100:.0f}%."
                })

        # Insight 4: Relationship between time
buffer and completion
        if (deadline_compliance and
"buffer_correlation" in deadline_compliance
and

deadline_compliance["buffer_correlation"] is
not None):

            correlation =
deadline_compliance["buffer_correlation"]
            logit =
deadline_compliance.get("buffer_logit")

            if abs(correlation) > 0.3:
                direction = "positive" if
correlation > 0 else "negative"

                insight = {
                    "type": "buffer",
                    "title": "Time Buffer
Impact",
                    "description": f"There is
a {direction} relationship between deadline
buffer and task completion."
                }

                if direction == "positive":
                    insight["description"] +=
" Tasks with more time tend to be completed
more often."
                else:
                    insight["description"] +=
" Tasks with less time actually tend to be
completed more often."

                if logit and
logit["significant"]:
                    insight["description"] +=
" This finding is statistically significant."

                insights.append(insight)

        # Insight 5: Identify overdue tasks
        overdue_tasks = [note for note in
self.notes if note.status == "todo" and
note.is_overdue()]
        if overdue_tasks:
            insights.append({
                "type": "overdue",
                "title": "Overdue Tasks",
                "description": f"You have
{len(overdue_tasks)} overdue tasks that need
attention.",
                "tasks": [(note.title,
```

```
note.category) for note in overdue_tasks[:3]]
            })

        return insights

    def get_deadline_recommendation(self,
note_content, category=None):
        if not self.notes:
            return {"error": "No historical
data available"}

        cat_notes = [note for note in
self.notes if note.category == category]
        if not cat_notes:
            return {"error": f"No historical
data for category '{category}'"}

        completed_notes = [note for note in
cat_notes
                           if note.status ==
"done" and note.completion_time() is not None]

        if not completed_notes:
            return {
                "category": category,
                "error": "No completed notes
in this category to base recommendation on"
            }

        completion_times =
[note.completion_time() for note in
completed_notes]
        avg_completion_time =
np.mean(completion_times)
        median_completion_time =
np.median(completion_times)
        std_completion_time =
np.std(completion_times) if
len(completion_times) > 1 else
avg_completion_time * 0.2

        word_count = len(note_content.split())

        word_counts =
[len(note.content.split()) for note in
completed_notes]

        try:
            slope, intercept, r_value,
p_value, std_err =
stats.linregress(word_counts,
completion_times)

            regression_valid = p_value < 0.05
and r_value ** 2 >= 0.3
            regression_estimate = slope *
word_count + intercept if regression_valid
else None

            regression_stats = {
                "slope": slope,
                "intercept": intercept,
                "r_squared": r_value ** 2,
                "p_value": p_value,
                "std_err": std_err,
                "valid": regression_valid
            }
        except:
            regression_valid = False
            regression_estimate = None
            regression_stats = None

        time_of_day_effect = 1.0
        try:
            time_periods = []
            for note in completed_notes:
                try:
                    create_date =
datetime.strptime(note.date, "%Y-%m-%d
%H:%M:%S")
                    hour = create_date.hour
                    if 5 <= hour < 12:
                        period = "morning"
                    elif 12 <= hour < 17:
                        period = "afternoon"
                    elif 17 <= hour < 22:
                        period = "evening"
                    else:
                        period = "night"

time_periods.append(period)
                except:

time_periods.append("unknown")

            time_df = pd.DataFrame({
                'time_of_day': time_periods,
                'hours_per_word': [time /
max(1, len(note.content.split()))
                                   for note,
time in zip(completed_notes,
completion_times)]
            })

            time_efficiency =
time_df.groupby('time_of_day')['hours_per_word
'].mean().to_dict()

            current_hour = datetime.now().hour
            if 5 <= current_hour < 12:
                current_period = "morning"
            elif 12 <= current_hour < 17:
                current_period = "afternoon"
            elif 17 <= current_hour < 22:
                current_period = "evening"
            else:
                current_period = "night"

            if current_period in
time_efficiency and "morning" in
time_efficiency:
                baseline =
time_efficiency["morning"]
                current_efficiency =
time_efficiency[current_period]
                if baseline > 0:
                    time_of_day_effect =
current_efficiency / baseline

            time_of_day_info = {
                "current_period":
current_period,
                "efficiency_factors":
time_efficiency,
                "adjustment_factor":
time_of_day_effect
            }
        except:
            time_of_day_effect = 1.0
            time_of_day_info = None

        estimates = []
        weights = []

        # 1. Simple average completion time
        estimates.append(avg_completion_time)
        weights.append(0.2)

        # 2. Regression-based estimate if
valid
        if regression_valid and
regression_estimate is not None and
regression_estimate > 0:

estimates.append(regression_estimate)
            weights.append(0.5 *
regression_stats["r_squared"])

        # 3. Word-count proportional estimate
        if word_counts and completion_times:
```

```
            avg_time_per_word =
sum(completion_times) / max(1,
sum(word_counts))
            word_based_estimate = word_count *
avg_time_per_word

estimates.append(word_based_estimate)
            weights.append(0.3)

        if estimates and sum(weights) > 0:
            weighted_estimate = sum(e * w for
e, w in zip(estimates, weights)) /
sum(weights)
        else:
            weighted_estimate =
avg_completion_time

        adjusted_estimate = weighted_estimate
* time_of_day_effect

        safety_factor = 1.0
        confidence_level = 0.8

        if len(completed_notes) >= 5:
            safety_factor =
stats.norm.ppf(confidence_level)
        else:
            safety_factor = 1.5

        recommended_hours = adjusted_estimate
+ (safety_factor * std_completion_time)

        confidence_levels = [0.5, 0.7, 0.8,
0.9, 0.95]
        confidence_intervals = {}

        for conf_level in confidence_levels:
            if len(completed_notes) > 1:
                t_value = stats.t.ppf((1 +
conf_level) / 2, len(completed_notes) - 1)

                margin = t_value *
std_completion_time * math.sqrt(1 + 1 /
len(completed_notes))
                lower_bound = max(0,
adjusted_estimate - margin)
                upper_bound =
adjusted_estimate + margin

                lower_date = datetime.now() +
timedelta(hours=lower_bound)
                upper_date = datetime.now() +
timedelta(hours=upper_bound)


confidence_intervals[str(int(conf_level *
100)) + "%"] = {
                    "lower hours":
round(lower_bound, 1),
                    "upper_hours":
round(upper_bound, 1),
                    "lower_date":
lower_date.strftime("%Y-%m-%d %H:%M:%S"),
                    "upper date":
upper_date.strftime("%Y-%m-%d %H:%M:%S")
                }
            else:

confidence_intervals[str(int(conf_level *
100)) + "%"] = {
                    "lower_hours":
round(adjusted_estimate * 0.5, 1),
                    "upper_hours":
round(adjusted_estimate * 2.0, 1),
                    "lower_date":
(datetime.now() +
timedelta(hours=adjusted_estimate *
0.5)).strftime(
                        "%Y-%m-%d %H:%M:%S"),
                    "upper_date":
(datetime.now() +
timedelta(hours=adjusted_estimate *
2.0)).strftime(
                        "%Y-%m-%d %H:%M:%S")
                }

        if len(completed_notes) > 1:
            z score = (recommended hours -
adjusted_estimate) / (
                        std_completion_time *
math.sqrt(1 + 1 / len(completed_notes)))
            completion_probability =
stats.t.cdf(z_score, len(completed_notes) - 1)
            completion_probability =
min(max(completion_probability, 0.5), 0.99)
        else:
            completion_probability = 0.8

        recommended_hours =
math.ceil(recommended_hours)

        deadline_date = datetime.now() +
timedelta(hours=recommended_hours)

        deadline_str =
deadline_date.strftime("%Y-%m-%d %H:%M:%S")

        return {
            "category": category,
            "word_count": word_count,
            "avg_completion_time":
avg_completion_time,
            "median_completion_time":
median_completion_time,
            "regression_stats":
regression_stats,
            "time_of_day_info":
time_of_day_info,
            "estimated_hours": {
                "simple_avg":
avg_completion_time,
                "regression":
regression_estimate,
                "word_based":
word_based_estimate if 'word_based_estimate'
in locals() else None,
                "weighted": weighted_estimate,
                "time_adjusted":
adjusted_estimate
            },
            "recommended_hours":
recommended_hours,
            "recommended_deadline":
deadline_str,
            "completion_probability":
round(completion_probability * 100, 1),
            "confidence_intervals":
confidence intervals,
            "notes_analyzed":
len(completed_notes),
            "confidence":
min(len(completed_notes) / 10, 0.9),
            "explanation": {
                "methods used": [
                    "Historical average
completion time",
                    "Regression analysis" if
regression_valid else None,
                    "Word count proportional
estimate",
                    "Time of day adjustment"
                ],
                "regression_valid":
regression_valid,
                "time_of_day_factor":
time_of_day_effect,
                "probability_explanation":
f"There is approximately a
{round(completion_probability * 100)}%
probability that you will complete this note
```

```
by the recommended deadline based on your
historical completion patterns."
            }
        }
```

note_category_predictor.py:

```
import math
import re
import numpy as np
from collections import Counter, defaultdict


class BaseCategoryPredictor:
    """Base class with common preprocessing
and utility methods"""

    def __init__(self):
        self.stopwords = {'the', 'and', 'a',
'to', 'of', 'in', 'i', 'is', 'that', 'it',
'for', 'you', 'this', 'with',
                          'on', 'be', 'are',
'as', 'at', 'have', 'from', 'or', 'not', 'by',
'but', 'what', 'all', 'can',
                          'was'}
        self.method_name = "Base"

    def preprocess_text(self, text):
        text = text.lower()
        text = re.sub(r'[^\w\s]', ' ', text)
        text = re.sub(r'\d+', ' ', text)
        words = text.split()
        words = [word for word in words if
word not in self.stopwords and len(word) > 2]
        return words

    def get_word_frequencies(self, words):
        return Counter(words)

    def predict_category(self, content,
categorized_notes):
        raise NotImplementedError("Subclasses
must implement this method")

class
TFIDFCosinePredictor(BaseCategoryPredictor):

    def __init__(self):
        super().__init__()
        self.method_name = "TF-IDF Cosine
Similarity"

    def calculate_tf_idf(self, note_words,
all_notes_words):
        word_freqs =
self.get_word_frequencies(note_words)

        doc_freq = {}
        for word in set(note_words):
            doc_freq[word] = sum(1 for
note_words_list in all_notes_words if word in
note_words_list)

        tf_idf = {}
        total docs = len(all notes words)
        for word, freq in word_freqs.items():
            tf = freq / max(1,
len(note_words))
            idf = math.log(total_docs / max(1,
doc_freq[word]))
            tf_idf[word] = tf * idf

        return tf_idf

    def cosine_similarity(self, vec1, vec2):
        all_words = set(list(vec1.keys()) +
list(vec2.keys()))

        dot_product = sum(vec1.get(word, 0) *
vec2.get(word, 0) for word in all_words)
        mag1 = math.sqrt(sum(vec1.get(word, 0)
** 2 for word in all words))
        mag2 = math.sqrt(sum(vec2.get(word, 0)
** 2 for word in all_words))

        if mag1 * mag2 == 0:
            return 0

        return dot_product / (mag1 * mag2)

    def predict_category(self, content,
categorized_notes):
        current_words =
self.preprocess_text(content)

        all_notes_words = []
        category_words_map = {}

        for note in categorized_notes:
            words =
self.preprocess_text(note.content)
            all_notes_words.append(words)

            if note.category not in
category_words_map:

category_words_map[note.category] = []

category_words_map[note.category].extend(words
)

        all_notes_words.append(current_words)

        current_tfidf =
self.calculate_tf_idf(current_words,
all_notes_words)

        category_tfidf = {}
        for category, words in
category_words_map.items():
            category_tfidf[category] =
self.calculate_tf_idf(words, all_notes_words)

        similarities = {}
        for category, tfidf in
category_tfidf.items():
            similarities[category] =
self.cosine_similarity(current_tfidf, tfidf)

        if not similarities:
            return ""

        best_category =
max(similarities.items(), key=lambda x:
x[1])[0]
        return best_category, similarities


class
NaiveBayesPredictor(BaseCategoryPredictor):

    def __init__(self):
        super().  init  ()
        self.method_name = "Naive Bayes"
        self.alpha = 1.0

    def train(self, categorized_notes):
        self.class_word_counts =
defaultdict(Counter)
        self.class_counts = Counter()
        self.vocabulary = set()

        for note in categorized_notes:
            category = note.category
            words =
self.preprocess_text(note.content)

            self.class_counts[category] += 1
```

```
self.class_word_counts[category].update(words)
            self.vocabulary.update(words)

        total_notes =
sum(self.class_counts.values())
        self.class_priors = {category: count /
total_notes
                             for category,
count in self.class_counts.items()}

    def predict_category(self, content,
categorized_notes):
        self.train(categorized_notes)

        if not self.class_counts:
            return "", {}

        words = self.preprocess_text(content)

        scores = {}
        for category in self.class_counts:
            score =
math.log(self.class_priors[category])

            total_words =
sum(self.class_word_counts[category].values())
            vocab_size = len(self.vocabulary)

            for word in words:
                if word not in
self.vocabulary:
                    continue

                word_count =
self.class_word_counts[category].get(word, 0)
                word_prob = (word_count +
self.alpha) / (total_words + self.alpha *
vocab_size)
                score += math.log(word_prob)

            scores[category] = score

        if scores:
            max_score = max(scores.values())
            exp_scores = {cat: math.exp(score
- max_score) for cat, score in scores.items()}
            total = sum(exp_scores.values())
            normalized_scores = {cat: score /
total for cat, score in exp_scores.items()}
            best_category =
max(normalized_scores.items(), key=lambda x:
x[1])[0]
            return best_category,
normalized_scores

        return "", {}


class
JaccardSimilarityPredictor(BaseCategoryPredict
or):

    def __init__(self):
        super().__init__()
        self.method_name = "Jaccard
Similarity"

    def jaccard_similarity(self, set1, set2):
        if not set1 or not set2:
            return 0
        intersection =
len(set1.intersection(set2))
        union = len(set1.union(set2))
        return intersection / union

    def predict_category(self, content,
categorized_notes):
        current_words =
set(self.preprocess_text(content))
        category_words = defaultdict(set)
        for note in categorized_notes:
            category = note.category
            words =
set(self.preprocess_text(note.content))

category_words[category].update(words)

        similarities = {}
        for category, words in
category_words.items():
            similarities[category] =
self.jaccard_similarity(current_words, words)

        if not similarities:
            return "", {}

        best_category =
max(similarities.items(), key=lambda x:
x[1])[0]
        return best_category, similarities


class
FrequencyDistancePredictor(BaseCategoryPredict
or):

    def __init__(self,
distance_metric="euclidean"):
        super().__init__()
        self.distance_metric = distance_metric
        self.method_name = f"Frequency
{distance_metric.capitalize()} Distance"

    def calculate_distance(self, vec1, vec2,
metric):
        all_words =
sorted(set(list(vec1.keys()) +
list(vec2.keys())))

        v1 = np.array([vec1.get(word, 0) for
word in all_words])
        v2 = np.array([vec2.get(word, 0) for
word in all_words])

        if metric == "euclidean":
            return np.sqrt(np.sum((v1 - v2) **
2))
        elif metric == "manhattan":
            return np.sum(np.abs(v1 - v2))
        elif metric == "chebyshev":
            return np.max(np.abs(v1 - v2))
        else:
            raise ValueError(f"Unknown
distance metric: {metric}")

    def predict_category(self, content,
categorized notes):
        current_words =
self.preprocess_text(content)
        current_freq =
self.get_word_frequencies(current_words)

        total = sum(current_freq.values())
        if total > 0:
            current_freq = {word: count /
total for word, count in current_freq.items()}

        category_freqs = defaultdict(Counter)
        category_total_words =
defaultdict(int)

        for note in categorized_notes:
            category = note.category
            words =
self.preprocess_text(note.content)
            freqs =
self.get_word_frequencies(words)

category_freqs[category].update(freqs)
```

```
            category_total_words[category] +=
sum(freqs.values())

        for category in category_freqs:
            total =
category_total_words[category]
            if total > 0:
                category freqs[category] =
{word: count / total

for word, count in
category_freqs[category].items()}

        distances = {}
        for category, freqs in
category_freqs.items():
            distances[category] =
self.calculate_distance(current_freq, freqs,
self.distance_metric)

        similarities = {}
        if distances:
            max_dist = max(distances.values())
if distances else 1
            for category, dist in
distances.items():
                similarities[category] = 1 -
(dist / max_dist)

        if not similarities:
            return "", {}

        best_category =
max(similarities.items(), key=lambda x:
x[1])[0]
        return best_category, similarities


class LSAPredictor(BaseCategoryPredictor):
    def __init__(self, n_components=100):
        super().__init__()
        self.method_name = f"LSA
({n_components} components)"
        self.n_components = n_components
        self.vectorizer = None
        self.U = None
        self.Sigma = None
        self.VT = None
        self.terms = []
        self.category_vectors = {}

    def compute_tf_idf_matrix(self, texts):
        vocab_counter = Counter()
        for words in texts:
            vocab_counter.update(set(words))

        vocab = sorted(vocab_counter.keys())
        word2idx = {word: idx for idx, word in
enumerate(vocab)}
        N = len(texts)

        idf = {}
        for word in vocab:
            df = vocab counter[word]
            idf[word] = math.log(N / (1 + df))
+ 1

        rows = []
        for words in texts:
            tf = Counter(words)
            doc_vector = np.zeros(len(vocab))
            for word, count in tf.items():
                if word in word2idx:
                    tf_val = count /
len(words)
                    doc_vector[word2idx[word]]
= tf_val * idf[word]
            rows.append(doc_vector)

        return np.array(rows), vocab

    def truncated_svd(self, X, k):
        U, s, VT = np.linalg.svd(X,
full_matrices=False)
        U_k = U[:, :k]
        s_k = s[:k]
        VT_k = VT[:k, :]
        return U k, s k, VT k

    def document_vector(self, words, vocab,
VT_k, s_k):
        word2idx = {word: idx for idx, word in
enumerate(vocab)}
        vec = np.zeros(len(vocab))
        tf = Counter(words)
        for word, count in tf.items():
            if word in word2idx:
                vec[word2idx[word]] = count /
len(words)

        tfidf_vec = vec
        term_projection = np.dot(tfidf_vec,
VT_k.T) / s_k
        return term_projection

    def predict_category(self, content,
categorized_notes):
        all_texts =
[self.preprocess_text(note.content) for note
in categorized_notes]
        all_categories = [note.category for
note in categorized_notes]

        if not all_texts:
            return "", {}

        X, vocab =
self.compute_tf_idf_matrix(all_texts)

        k = min(self.n_components,
min(X.shape) - 1)
        U_k, s_k, VT_k = self.truncated_svd(X,
k)

        category_vectors = defaultdict(list)
        for i, category in
enumerate(all_categories):

category_vectors[category].append(U_k[i])

        avg_category_vectors = {}
        for category, vectors in
category_vectors.items():
            avg_category_vectors[category] =
np.mean(vectors, axis=0)

        current_words =
self.preprocess text(content)
        current_vector =
self.document_vector(current_words, vocab,
VT_k, s_k)

        similarities = {}
        for category, vec in
avg_category_vectors.items():
            norm = np.linalg.norm(vec) *
np.linalg.norm(current_vector)
            if norm == 0:
                similarities[category] = 0
            else:
                similarities[category] =
np.dot(vec, current_vector) / norm

        if not similarities:
            return "", {}

        best_category =
max(similarities.items(), key=lambda x:
x[1])[0]
        return best_category, similarities
```

```
class MultiMethodCategoryPredictor:

    def __init__(self, glove_path=None):
        self.predictors = [
            TFIDFCosinePredictor(),
            NaiveBayesPredictor(),
            JaccardSimilarityPredictor(),

FrequencyDistancePredictor(distance_metric="eu
clidean"),

FrequencyDistancePredictor(distance_metric="ma
nhattan"),
            LSAPredictor(n_components=100)
        ]

    def predict_category(self, content,
categorized_notes):
        results = {}

        for predictor in self.predictors:
            best_category, scores =
predictor.predict_category(content,
categorized_notes)
            results[predictor.method_name] = {
                'prediction': best_category,
                'confidence_scores': scores
            }

        return results

    def get_consensus_prediction(self,
results):
        predictions = [result['prediction']
for result in results.values()]
        counter = Counter(predictions)

        if not counter:
            return ""

        return counter.most_common(1)[0][0]
```

socket_service.py:

```
import socket
import json
import threading
import time


class NoteSocketService:
    def __init__(self, host='localhost',
port=8080):
        self.host = host
        self.port = port
        self.connected = False
        self.reconnect_attempts = 0
        self.max_reconnect_attempts = 5
        self.socket = None
        self.receive_callback = None
        self.lock = threading.Lock()

    def connect(self):
        # Establish connection to the C++
socket server
        try:
            self.socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.socket.connect((self.host,
self.port))
            self.connected = True
            self.reconnect_attempts = 0

            receive_thread =
threading.Thread(target=self._receive_messages
)
            receive_thread.daemon = True
            receive_thread.start()

            return True
        except Exception as e:
            print(f"Connection error: {e}")
            self.connected = False
            return False

    def disconnect(self):
        if self.socket:
            try:
                self.socket.close()
            except:
                pass
            self.socket = None
        self.connected = False

    def _ensure_connection(self):
        if not self.connected:
            if self.reconnect_attempts <
self.max_reconnect_attempts:
                self.reconnect_attempts += 1
                return self.connect()
            return False
        return True

    def send_note(self, note,
operation="save"):
        with self.lock:
            if not self._ensure_connection():
                raise ConnectionError("Failed
to connect to notes server")

            note_data = {
                "operation": operation,
                "note": {
                    "id": note.id,
                    "title": note.title,
                    "content": note.content,
                    "date": note.date,
                    "category": note.category,
                    "status": note.status,
                    "deadline": note.deadline,
                    "completion_date":
note.completion_date
                }
            }

            try:
                message =
json.dumps(note_data) + "\n"

self.socket.sendall(message.encode('utf-8'))
                return True
            except Exception as e:
                print(f"Error sending Note:
{e}")
                self.connected = False
                return False

    def delete_note(self, note_id):
        with self.lock:
            if not self._ensure_connection():
                raise ConnectionError("Failed
to connect to notes server")

            delete_data = {
                "operation": "delete",
                "note_id": note_id
            }

            try:
                message =
json.dumps(delete_data) + "\n"

self.socket.sendall(message.encode('utf-8'))
                return True
            except Exception as e:
                print(f"Error deleting Note:
{e}")
```

```
                self.connected = False
                return False

    def request_notes(self):
        with self.lock:
            if not self._ensure_connection():
                raise ConnectionError("Failed
to connect to notes server")

            request_data = {
                "operation": "list"
            }

            try:
                message =
json.dumps(request_data) + "\n"

self.socket.sendall(message.encode('utf-8'))
                return True
            except Exception as e:
                print(f"Error requesting
notes: {e}")
                self.connected = False
                return False

    def get_note(self, note_id):
        with self.lock:
            if not self._ensure_connection():
                raise ConnectionError("Failed
to connect to notes server")

            request_data = {
                "operation": "get",
                "note_id": note_id
            }

            try:
                message =
json.dumps(request_data) + "\n"

self.socket.sendall(message.encode('utf-8'))
                return True
            except Exception as e:
                print(f"Error getting Note:
{e}")
                self.connected = False
                return False

    def set_receive_callback(self, callback):
        self.receive_callback = callback

    def _receive_messages(self):
        buffer = ""

        while self.connected:
            try:
                data = self.socket.recv(4096)
                if not data:
                    self.connected = False
                    break

                buffer += data.decode('utf-8')

                while '\n' in buffer:
                    message, buffer =
buffer.split('\n', 1)

                    try:
                        response =
json.loads(message)
                        if
self.receive_callback:

self.receive_callback(response)
                    except
json.JSONDecodeError:
                        print(f"Error decoding
message: {message}")

            except Exception as e:
                print(f"Error receiving data:
{e}")
                self.connected = False
                break

        self.socket = None
```

analyzer_ui.py:

```
import tkinter as tk
from tkinter import messagebox, ttk
from ..Stats.note_analytics import
NotesAnalytics

import numpy as np
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg

class AnalyzerUI:
    def __init__(self, root):
        self.root = root
        self.analytics = NotesAnalytics()

    def update_analytics_data(self, notes):
        self.analytics.set_notes(notes)
        self.notes = notes

    def show_summary_statistics(self):
        stats =
self.analytics.get_summary_statistics()

        if not stats:
            messagebox.showinfo("No Data",
"Not enough data to generate statistics.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Summary
Statistics")
        report_window.geometry("600x400")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Notes Summary
Statistics", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        basic_frame = ttk.LabelFrame(frame,
text="Basic Statistics")
        basic_frame.pack(fill=tk.X, pady=10)

        ttk.Label(basic_frame, text=f"Total
Notes:
{stats['total_notes']}").pack(anchor="w",
padx=10, pady=2)

        status_frame = ttk.LabelFrame(frame,
text="Status Distribution")
        status_frame.pack(fill=tk.X, pady=10)

        for status, count in
stats['status counts'].items():
            ttk.Label(status_frame,
text=f"{status.capitalize()}:
{count}").pack(anchor="w", padx=10, pady=2)

        if stats['completion_rate'] is not
None:
            completion_frame =
ttk.LabelFrame(frame, text="Completion
Statistics")
            completion_frame.pack(fill=tk.X,
pady=10)

            ttk.Label(completion_frame,
                      text=f"Completion Rate:
{stats['completion_rate'] *
```

```
100:.1f}%").pack(anchor="w", padx=10, pady=2)

            if stats['avg completion time'] is
not None:
                ttk.Label(completion_frame,
                          text=f"Avg.
Completion Time:
{stats['avg completion time']:.2f}
hours").pack(anchor="w",

padx=10, pady=2)

            if stats['on_time_rate'] is not
None:
                ttk.Label(completion_frame,
                          text=f"On-time Rate:
{stats['on_time_rate'] *
100:.1f}%").pack(anchor="w", padx=10, pady=2)

        if stats['category_counts']:
            category_frame =
ttk.LabelFrame(frame, text="Top Categories")
            category_frame.pack(fill=tk.X,
pady=10)

            sorted_categories =
sorted(stats['category_counts'].items(),
key=lambda x: x[1], reverse=True)
            for category, count in
sorted_categories[:5]:
                ttk.Label(category_frame,
text=f"{category}: {count}").pack(anchor="w",
padx=10, pady=2)

    def show_category_analysis(self):
        stats =
self.analytics.get_category_stats()

        if not stats:
            messagebox.showinfo("No Data",
"Not enough data to generate category
statistics.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Category
Analysis")
        report_window.geometry("700x500")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Category
Analysis", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        notebook = ttk.Notebook(frame)
        notebook.pack(fill=tk.BOTH,
expand=True)

        for category, cat_stats in
stats.items():
            cat frame = ttk.Frame(notebook,
padding="10")
            notebook.add(cat_frame,
text=category)

            ttk.Label(cat_frame, text=f"Total
Notes: {cat_stats['count']}", font=('Arial',
11)).pack(anchor="w", pady=2)

            status_frame =
ttk.LabelFrame(cat_frame, text="Status
Distribution")
            status_frame.pack(fill=tk.X,
pady=10)

            for status, count in
cat_stats['status_distribution'].items():
                percentage = count /
cat_stats['count'] * 100
                ttk.Label(status frame,
text=f"{status.capitalize()}: {count}
({percentage:.1f}%)").pack(anchor="w",

padx=10,

pady=2)

            if cat_stats['completion_rate'] is
not None:
                completion_frame =
ttk.LabelFrame(cat_frame, text="Completion
Statistics")

completion_frame.pack(fill=tk.X, pady=10)

                ttk.Label(completion_frame,
                          text=f"Completion
Rate: {cat_stats['completion_rate'] *
100:.1f}%").pack(anchor="w", padx=10,

pady=2)

                if
cat_stats['avg_completion_time'] is not None:

ttk.Label(completion_frame,
                                  text=f"Avg.
Completion Time:
{cat_stats['avg_completion_time']:.2f}
hours").pack(
                                  anchor="w", padx=10,
pady=2)

ttk.Label(completion_frame,
                                  text=f"Median
Completion Time:
{cat_stats['median_completion_time']:.2f}
hours").pack(
                                  anchor="w", padx=10,
pady=2)

                if cat_stats['on_time_rate']
is not None:

ttk.Label(completion_frame,
                                  text=f"On-time
Rate: {cat_stats['on_time_rate'] *
100:.1f}%").pack(anchor="w", padx=10,

pady=2)

    def show_completion_speed(self):
        speed_data =
self.analytics.get_completion_speed()

        if not speed_data:
            messagebox.showinfo("No Data",
"Not enough completed notes to analyze
completion speed.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Completion Speed
Analysis")
        report_window.geometry("700x500")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Completion
Speed Analysis", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        if "correlation" in speed_data:
            correlation_frame =
ttk.LabelFrame(frame, text="Word Count vs.
```

```
Completion Time")
            correlation_frame.pack(fill=tk.X,
pady=10)

            corr = speed_data["correlation"]
            corr_strength = "strong" if
abs(corr) > 0.7 else "moderate" if abs(corr) >
0.3 else "weak"
            corr_direction = "positive" if
corr > 0 else "negative"

            ttk.Label(correlation_frame,
                      text=f"There is a
{corr_strength} {corr_direction} correlation
({corr:.2f}) between word count and completion
time.").pack(
                anchor="w", padx=10, pady=2)

            if "regression" in speed_data:
                reg = speed_data["regression"]
                ttk.Label(correlation_frame,
                          text=f"Predictive
model: Time (hrs) = {reg['slope']:.4f} × Words
+ {reg['intercept']:.2f}").pack(
                    anchor="w", padx=10,
pady=2)
                ttk.Label(correlation_frame,
                          text=f"Model
explains {reg['r_squared'] * 100:.1f}% of the
variation in completion times.").pack(
                    anchor="w", padx=10,
pady=2)

        if "category_efficiency" in speed_data
and speed_data["category_efficiency"]:
            cat_frame = ttk.LabelFrame(frame,
text="Category Efficiency (Hours per Word)")
            cat_frame.pack(fill=tk.X, pady=10)

            sorted_cats =
sorted(speed_data["category_efficiency"].items
(), key=lambda x: x[1])

            for cat, speed in sorted_cats:
                ttk.Label(cat_frame,
text=f"{cat}: {speed:.5f}
hours/word").pack(anchor="w", padx=10, pady=2)

            best_cat, best_time =
sorted_cats[0]
            worst_cat, worst_time =
sorted_cats[-1]

            diff_factor = worst_time /
best_time

            ttk.Label(cat_frame,
                      text=f"\nYou complete
'{best_cat}' notes {diff_factor:.1f}x faster
than '{worst_cat}' notes.").pack(
                anchor="w", padx=10, pady=2)

        if "time_of_day_efficiency" in
speed data and
speed_data["time_of_day_efficiency"]:
            time_frame = ttk.LabelFrame(frame,
text="Productivity by Time of Day")
            time_frame.pack(fill=tk.X,
pady=10)

            sorted_times =
sorted(speed_data["time_of_day_efficiency"].it
ems(), key=lambda x: x[1])

            for time_period, speed in
sorted_times:
                ttk.Label(time_frame,
text=f"{time_period.capitalize()}: {speed:.5f}
hours/word").pack(anchor="w",
padx=10, pady=2)

            if
speed_data.get("time_of_day_anova"):
                anova =
speed_data["time_of_day_anova"]
                if anova["significant"]:
                    ttk.Label(time_frame,
                              text="\nThis
difference across time periods is
statistically significant.").pack(
                        anchor="w", padx=10,
pady=2)
                else:
                    ttk.Label(time_frame,
                              text="\nThis
difference is not statistically
significant.").pack(anchor="w", padx=10,

pady=2)

    def show_deadline_compliance(self):
        deadline_data =
self.analytics.get_deadline_compliance()

        if not deadline_data:
            messagebox.showinfo("No Data",
"Not enough notes with deadlines to analyze
compliance.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Deadline
Compliance Analysis")
        report_window.geometry("700x500")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Deadline
Compliance Analysis", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        overall_frame = ttk.LabelFrame(frame,
text="Overall Statistics")
        overall_frame.pack(fill=tk.X, pady=10)

        ttk.Label(overall_frame,
                  text=f"Completion Rate:
{deadline_data['completion_rate'] *
100:.1f}%").pack(anchor="w", padx=10,

pady=2)
        ttk.Label(overall_frame,
                  text=f"On-time Rate:
{deadline_data['on_time_rate'] *
100:.1f}%").pack(anchor="w", padx=10, pady=2)

        if
deadline_data.get('avg_hours_to_deadline') is
not None:
            avg_hours =
deadline data['avg hours to deadline']
            if avg_hours >= 0:
                ttk.Label(overall_frame,
                          text=f"Average:
{abs(avg_hours):.1f} hours before
deadline").pack(anchor="w", padx=10, pady=2)
            else:
                ttk.Label(overall_frame,
                          text=f"Average:
{abs(avg_hours):.1f} hours after
deadline").pack(anchor="w", padx=10, pady=2)

        if
deadline_data.get('category_stats'):
            cat_frame = ttk.LabelFrame(frame,
text="Performance by Category")
            cat_frame.pack(fill=tk.X, pady=10)
```

```
            sorted_cats =
sorted(deadline_data['category_stats'].items()
,
                                   key=lambda x:
x[1]['on_time_rate'], reverse=True)

            for cat, stats in sorted_cats:
                cat_on_time =
stats['on_time_rate'] * 100
                cat_avg_hours =
stats['avg_adherence']

                if cat_avg_hours >= 0:
                    time_text =
f"{abs(cat_avg_hours):.1f} hours before
deadline"
                else:
                    time_text =
f"{abs(cat_avg_hours):.1f} hours after
deadline"

                ttk.Label(cat_frame,
                          text=f"{cat}:
{cat_on_time:.1f}% on-time, avg
{time_text}").pack(anchor="w", padx=10,
pady=2)

        if
deadline_data.get('buffer_correlation') is not
None:
            buffer_frame =
ttk.LabelFrame(frame, text="Deadline Buffer
Impact")
            buffer_frame.pack(fill=tk.X,
pady=10)

            corr =
deadline_data['buffer_correlation']
            corr_strength = "strong" if
abs(corr) > 0.7 else "moderate" if abs(corr) >
0.3 else "weak"
            corr_direction = "positive" if
corr > 0 else "negative"

            if corr_direction == "positive":
                interpretation = "Tasks with
more buffer time are more likely to be
completed."
            else:
                interpretation = "Tasks with
less buffer time are actually more likely to
be completed."

            ttk.Label(buffer_frame,
                      text=f"There is a
{corr_strength} {corr_direction} correlation
({corr:.2f}) between buffer time and
completion.").pack(
                anchor="w", padx=10, pady=2)
            ttk.Label(buffer_frame,
text=interpretation).pack(anchor="w", padx=10,
pady=2)

        if deadline_data.get('buffer_logit'):
            logit_frame =
ttk.LabelFrame(frame, text="Logistic
Regression Analysis")
            logit_frame.pack(fill=tk.X,
pady=10)

            logit_data =
deadline_data['buffer_logit']
            coefficient =
logit_data['coefficient']
            intercept =
logit_data['intercept']
            score = logit_data['score']
            significant =
logit_data['significant']
            coef_sign = "+" if coefficient > 0
else ""

            ttk.Label(logit_frame,
                      text=f"Probability of
completion = 1/(1+e^-({intercept:.3f}
{coef_sign}{coefficient:.3f}×hours))").pack(
                anchor="w", padx=10, pady=2)

            ttk.Label(logit_frame,
                      text=f"Model accuracy:
{score:.1%}").pack(anchor="w", padx=10,
pady=2)

            if coefficient > 0:
                coef_interp = "Each additional
hour of buffer time increases the odds of
completion."
            else:
                coef_interp = "Each additional
hour of buffer time decreases the odds of
completion."

            sig_text = "This relationship is
statistically significant." if significant
else "This relationship is not statistically
significant."

            ttk.Label(logit_frame,
text=coef_interp).pack(anchor="w", padx=10,
pady=2)
            ttk.Label(logit_frame,
text=sig_text).pack(anchor="w", padx=10,
pady=2)

            if abs(coefficient) > 0.001:
                try:
                    fig = Figure(figsize=(5,
2.5))
                    ax = fig.add_subplot(111)

                    max_hours = min(48, 10 /
abs(coefficient)) if coefficient != 0 else 48
                    x = np.linspace(0,
max_hours, 100)

                    prob = 1 / (1 + np.exp(-
(intercept + coefficient * x)))

                    ax.plot(x, prob)
                    ax.set_xlabel('Buffer Time
(hours)')
                    ax.set_ylabel('Probability
of Completion')
                    ax.set_title('Completion
Probability vs. Buffer Time')
                    ax.grid(True, linestyle='-
-', alpha=0.7)

                    ax.axhline(y=0.5,
color='r', linestyle='--', alpha=0.5)

                    if coefficient != 0:
                        x_at_half = -intercept
/ coefficient
                        if 0 <= x_at_half <=
max_hours:

ax.axvline(x=x_at_half, color='g',
linestyle='--', alpha=0.5)
                            ax.text(x_at_half
+ 0.5, 0.5, f'{x_at_half:.1f} hrs', color='g')

                    canvas =
FigureCanvasTkAgg(fig, master=logit_frame)
                    canvas.draw()

canvas.get_tk_widget().pack(fill=tk.BOTH,
expand=True, padx=10, pady=10)
```

```
                except Exception as e:
                    ttk.Label(logit frame,
text="(Visualization
unavailable)").pack(anchor="w", padx=10,
pady=2)

    def show_content_analysis(self):
        content_data =
self.analytics.get_content_analysis()

        if not content_data or not
content_data.get('length_stats'):
            messagebox.showinfo("No Data",
"Not enough data to analyze Note content.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Content
Analysis")
        report_window.geometry("700x500")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Note Content
Analysis", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        length_stats =
content_data.get('length_stats', {})
        if length_stats:
            length_frame =
ttk.LabelFrame(frame, text="Text Length
Statistics")
            length_frame.pack(fill=tk.X,
pady=10)

            ttk.Label(length_frame,
                      text=f"Average Length:
{length_stats['avg_length']:.1f}
words").pack(anchor="w", padx=10, pady=2)
            ttk.Label(length_frame,
                      text=f"Median Length:
{length_stats['median_length']:.1f}
words").pack(anchor="w", padx=10,

pady=2)

            if
length_stats.get('by_category'):
                ttk.Label(length_frame,
text="\nAverage Length by Category:",
font=('Arial', 10, 'bold')).pack(
                    anchor="w", padx=10,
pady=(10, 2))

                sorted_cats =
sorted(length_stats['by_category'].items(),

key=lambda x: x[1]['avg_length'],
reverse=True)

                for cat, cat_stats in
sorted_cats:
                    ttk.Label(length_frame,
                              text=f"{cat}:
{cat_stats['avg_length']:.1f}
words").pack(anchor="w", padx=10, pady=2)

        word_freq =
content_data.get('word_freq', {})
        if word_freq:
            word_frame = ttk.LabelFrame(frame,
text="Word Analysis by Category")
            word_frame.pack(fill=tk.X,
pady=10)

            notebook =
ttk.Notebook(word_frame)
            notebook.pack(fill=tk.BOTH,
expand=True, pady=10)

            for category, words_data in
word_freq.items():
                cat frame =
ttk.Frame(notebook, padding="10")
                notebook.add(cat_frame,
text=category)

                ttk.Label(cat_frame,
text="Most Common Words:", font=('Arial', 10,
'bold')).pack(anchor="w",

pady=(0, 5))

                common_words_text = ",
".join([f"{word} ({count})" for word, count in
words_data.get('top_words', [])])
                ttk.Label(cat_frame,
text=common_words_text,
wraplength=500).pack(anchor="w", pady=(0, 10))

                ttk.Label(cat_frame,
text="Most Distinctive Words:", font=('Arial',
10, 'bold')).pack(anchor="w",

pady=(0, 5))

                distinctive_words_text = ",
".join(
                    [f"{word} ({score:.2f})"
for word, score in
words_data.get('distinctive_words', [])])
                ttk.Label(cat_frame,
text=distinctive_words_text,
wraplength=500).pack(anchor="w")

    def show_actionable_insights(self):
        insights =
self.analytics.get_actionable_insights()

        if not insights:
            messagebox.showinfo("No Insights",
"Not enough data to generate actionable
insights.")
            return

        report_window = tk.Toplevel(self.root)
        report_window.title("Actionable
Insights")
        report_window.geometry("700x500")

        frame = ttk.Frame(report_window,
padding="20")
        frame.pack(fill=tk.BOTH, expand=True)

        ttk.Label(frame, text="Actionable
Insights", font=('Arial', 14,
'bold')).pack(pady=(0, 20))

        for i, insight in enumerate(insights):
            insight_frame =
ttk.LabelFrame(frame, text=f"{i + 1}.
{insight['title']}")
            insight_frame.pack(fill=tk.X,
pady=10)

            ttk.Label(insight_frame,
text=insight['description'],
wraplength=600).pack(anchor="w", padx=10,
pady=2)

            if insight.get('comparison'):
                ttk.Label(insight_frame,
text=insight['comparison'],
wraplength=600).pack(anchor="w", padx=10,
pady=2)
```

```
            if insight.get('type') ==
'overdue' and insight.get('tasks'):
                ttk.Label(insight_frame,
text="\nTop overdue tasks:", font=('Arial',
10, 'bold')).pack(anchor="w",

padx=10,

pady=(5, 2))

                for title, category in
insight['tasks']:
                    cat_text = f"
[{category}]" if category else ""
                    ttk.Label(insight_frame,
text=f"• {title}{cat_text}").pack(anchor="w",
padx=20, pady=2)
```

noteflow_ui.py:

```
import tkinter as tk
from tkinter import messagebox
from datetime import datetime

from ..Note.note import Note
from ..UI.predictor_ui import PredictorUI
from ..Transport.socket_service import
NoteSocketService

class NoteflowUI (PredictorUI):
    def __init__(self, root):
        # Initialize socket service and
connect to C++ backend
        self.socket_service =
NoteSocketService()

self.socket_service.set_receive_callback(self.
_handle_socket_response)
        self._connect_to_server()
        super().__init__(root)

    def _update_connection_status(self):
        if self.socket_service.connected:

self.connection_status.config(text="🟢
Connected", foreground="green")
        else:

self.connection_status.config(text="🔴
Disconnected", foreground="red")

    def _reconnect(self):
        if self.socket_service.connect():
            messagebox.showinfo("Connection",
"Connected to notes server")
            self._update_connection_status()
            self._request_notes_from_server()
        else:
            messagebox.showwarning("Connection
Failed", "Could not connect to notes server")
            self._update_connection_status()

    def _connect_to_server(self):
        if not self.socket_service.connect():
            messagebox.showwarning(
                "Connection Failed",
                "Could not connect to notes
server. Working in offline mode."
            )
        else:
            print("Connected to notes server")


    def _request_notes_from_server(self):
        if not self.socket_service.connected:
            messagebox.showwarning("Not
Connected", "Not connected to the notes
server")
            return

        try:

self.socket_service.request_notes()
        except ConnectionError as e:
            messagebox.showerror("Connection
Error", str(e))
            self._update_connection_status()

    def _handle_socket_response(self,
response):
        operation = response.get("operation")
        status = response.get("status")

        if status == "error":
            messagebox.showerror("Server
Error", response.get("message", "Unknown
error"))
            return

        if operation == "list":

self._process_received_notes(response.get("not
es", []))
        elif operation == "save":
            note_id = response.get("note_id")
            if self.current_note_index is not
None and 0 <= self.current_note_index <
len(self.notes):

self.notes[self.current_note_index].id =
note_id
                messagebox.showinfo("Saved",
"Note saved successfully!")
        elif operation == "delete":
            messagebox.showinfo("Deleted",
"Note deleted successfully!")
        elif operation == "get":
            note_data = response.get("Note")
            if note_data:

self._display_received_note(note_data)

    def _process_received_notes(self,
notes_data):
        self.notes = []

        for note_json in notes_data:
            note = Note(
                note_json.get("id", "0"),
                note_json.get("title",
"Untitled"),
                note_json.get("content", ""),
                note_json.get("date", ""),
                note_json.get("category", ""),
                note_json.get("status",
"normal"),
                note_json.get("deadline", ""),

note_json.get("completion_date", "")
            )
            note.id = note_json.get("id")
            self.notes.append(note)

        self._update_notes_listbox()

        if self.notes:

self.notes_listbox.selection_set(0)
            self._on_note_select(None)
        else:
            self.current_note_index = None
            self.title_var.set("")
            self.category_var.set("")
            self.status_var.set("normal")
            self.text_area.delete(1.0, tk.END)
            self.date_label.config(text="")

self.completed_date_label.config(text="")
```

```
self.deadline_date_label.config(text="")

        self._update_predict_button_state()


self.analyzer_window.update_analytics_data(sel
f.notes)

    def _display_received_note(self,
note_data):
        note = Note(
            note_data.get("id", "0"),
            note_data.get("title",
"Untitled"),
            note_data.get("content", ""),
            note_data.get("date", ""),
            note_data.get("category", ""),
            note_data.get("status", "normal"),
            note_data.get("deadline", ""),
            note_data.get("completion_date",
"")
        )
        note.id = note_data.get("id")

        for i, existing_note in
enumerate(self.notes):
            if existing_note.id == note.id:
                self.notes[i] = note
                self.current_note_index = i
                break
            else:
                self.notes.append(note)
                self.current_note_index =
len(self.notes) - 1

        self._update_notes_listbox()
        self.title_var.set(note.title)
        self.category_var.set(note.category)
        self.status_var.set(note.status)
        self.text_area.delete(1.0, tk.END)
        self.text_area.insert(tk.END,
note.content)

        if note.date:

self.date_label.config(text=f"Created:
{note.date}")
        else:
            self.date_label.config(text="")

        if note.deadline:

self.deadline_date_label.config(text=f"Deadlin
e: {note.deadline}")
        else:

self.deadline date label.config(text="")

        if note.status == "done":

self.completed_date_label.config(text=f"Compet
ed: {note.completion_date}")
        else:

self.completed_date_label.config(text="")

    def _create_new_note(self):
        new_note = Note(-1, "New Note", "",
datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
"", "normal", datetime.now().strftime("%Y-%m-
%d %H:%M:%S"), datetime.now().strftime("%Y-%m-
%d %H:%M:%S"))

        self.notes.insert(len(self.notes),
new_note)
        self.current_note_index =
len(self.notes) - 1

        self._update_notes_listbox()
        self.notes_listbox.selection_clear(0,
tk.END)
        self.notes listbox.selection set(0)
        self.title_var.set(new_note.title)
        self.category_var.set("")
        self.status_var.set("normal")
        self.text_area.delete(1.0, tk.END)
        self.date label.config(text=f"Created:
{new_note.date}")

self.deadline_date_label.config(text=f"Complet
ed: {new_note.deadline}")

self.completed_date_label.config(text=f"Comple
ted: {new_note.completion_date}")

        self.title_entry.focus_set()

        self._update_predict_button_state()

    def _save_current_note(self):
        if self.current_note_index is None or
not (0 <= self.current_note_index <
len(self.notes)):
            return

        note =
self.notes[self.current_note_index]
        note.id =
self.notes[self.current_note_index].id
        note.title = self.title_var.get()
        note.content = self.text_area.get(1.0,
tk.END).strip()
        note.category =
self.category_var.get()
        note.status = self.status_var.get()
        if note.status == "done":
            note.completion_date =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        note.date =
self.date_label.cget("text").replace("Created:
", "")

        if self.has_deadline_var.get():
            try:
                year =
int(self.deadline_year_var.get())
                month =
int(self.deadline_month_var.get())
                day =
int(self.deadline_day_var.get())
                hour =
int(self.deadline_hour_var.get())
                minute =
int(self.deadline_minute_var.get())

                if month < 1 or month > 12 or
day < 1 or day > 31 or hour < 0 or hour > 23
or minute < 0 or minute > 59:
                    raise ValueError("Invalid
date or time values")

                deadline_date = datetime(year,
month, day, hour, minute)
                note.deadline =
deadline_date.strftime("%Y-%m-%d %H:%M:%S")
            except ValueError as e:
                messagebox.showerror("Invalid
Date", f"Please enter a valid date and time:
{e}")
                return
        else:
            note.deadline = None

        try:

self.socket_service.send_note(note)

            self._update_notes_listbox()
        except Exception as e:
```

```
            messagebox.showerror("Save Error",
f"Error saving Note: {e}")

        self._update_predict_button_state()


self.analyzer_window.update_analytics_data(sel
f.notes)

    def _delete_current_note(self):
        if self.current_note_index is None or
not (0 <= self.current_note_index <
len(self.notes)):
            messagebox.showinfo("No Note
Selected", "Please select a Note to delete.")
            return

        if messagebox.askyesno("Confirm
Delete", "Are you sure you want to delete this
Note?"):
            if not
self.socket_service.connected:
                messagebox.showwarning("Not
Connected", "Cannot delete notes without
server connection.")
                return

            note =
self.notes[self.current_note_index]
            note_id = note.id

            if note_id is not None:
                try:

self.socket_service.delete_note(note_id)
                except Exception as e:

messagebox.showerror("Delete Error", f"Error
deleting Note: {e}")
                    return


self.notes.pop(self.current_note_index)

            if self.notes:
                new_index =
min(self.current_note_index, len(self.notes) -
1)
                self.current_note_index =
new_index
                self._update_notes_listbox()

                if
self.displayed_notes_indices:
                    try:
                        displayed_idx =
self.displayed_notes_indices.index(new_index)

self.notes_listbox.selection_set(displayed_idx
)
                    except ValueError:

self.notes_listbox.selection_set(0)

self.current_note_index =
self.displayed_notes_indices[0]

                    self._on_note_select(None)
                else:
                    self.current_note_index =
None
                    self._clear_editor()
            else:
                self.current_note_index = None
                self._clear_editor()
                self._update_notes_listbox()


self._update_predict_button_state()
```

notepad_interface.py:

```
import tkinter as tk
from tkinter import messagebox, ttk,
PhotoImage
from datetime import datetime

from ..Stats.note_category_predictor import
MultiMethodCategoryPredictor
from ..UI.analyzer_ui import AnalyzerUI

class NotepadInterface:
    def __init__(self, root):
        self.root = root
        self.root.title("Noteflow AI")
        self.root.geometry("1280x720")

        # icon =
PhotoImage(file='Frontend/UI/raw/NAI.png')
        # self.root.iconphoto(True, icon)

        self.notes = []
        self.current_note_index = None

        self.displayed_notes_indices = []

        self.status_colors = {
            "normal": "",
            "todo": "#ffe6e6",
            "done": "#e6ffe6"
        }

        self.category_predictor =
MultiMethodCategoryPredictor()

        self._create_ui()
        self._create_menu()

        self._request_notes_from_server()

    def _create_ui(self):
        self.left_frame = ttk.Frame(self.root,
padding="10")
        self.right_frame =
ttk.Frame(self.root, padding="10")

        self.left_frame.pack(side=tk.LEFT,
fill=tk.BOTH, expand=False, padx=5, pady=5)
        self.right_frame.pack(side=tk.RIGHT,
fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.note_list_label =
ttk.Label(self.left_frame, text="Your Notes")
        self.note_list_label.pack(anchor="w",
pady=(0, 5))

        self.search_var = tk.StringVar()
        self.search_var.trace("w",
self._filter_notes)
        self.search_entry =
ttk.Entry(self.left_frame,
textvariable=self.search_var)
        self.search_entry.pack(fill=tk.X,
pady=(0, 5))

        self.notes_listbox = tk.Listbox(
            self.left_frame,
            width=30,
            height=25,
            activestyle='none',
            selectbackground='#a6a6a6',
            selectforeground='black'
        )
        self.notes_listbox.pack(fill=tk.BOTH,
expand=True)

self.notes_listbox.bind('<<ListboxSelect>>',
self._on_note_select)

        self.add_note_button =
```

```
ttk.Button(self.left_frame, text="+ New Note",
command=self._create_new_note)
        self.add note button.pack(fill=tk.X,
pady=(5, 0))

        self.delete_note_button =
ttk.Button(self.left_frame, text="🗑 Delete
Note", command=self._delete_current_note)

self.delete_note_button.pack(fill=tk.X,
pady=(5, 0))

        self.connection_status =
ttk.Label(self.left_frame, text="⬤
Disconnected", foreground="red")
        self.connection status.pack(fill=tk.X,
pady=(5, 0))

        self.reconnect_button =
ttk.Button(self.left_frame, text="🔄
Reconnect", command=self._reconnect)
        self.reconnect_button.pack(fill=tk.X,
pady=(5, 0))

        self.title_frame =
ttk.Frame(self.right_frame)
        self.title_frame.pack(fill=tk.X,
pady=(0, 5))

        self.title_label =
ttk.Label(self.title_frame, text="Title:")
        self.title_label.pack(side=tk.LEFT,
padx=(0, 5))

        self.title_var = tk.StringVar()
        self.title_entry =
ttk.Entry(self.title_frame,
textvariable=self.title_var)
        self.title_entry.pack(side=tk.LEFT,
fill=tk.X, expand=True)

        self.category_frame =
ttk.Frame(self.right_frame)
        self.category_frame.pack(fill=tk.X,
pady=(0, 5))

        self.category_label =
ttk.Label(self.category frame,
text="Category:")
        self.category_label.pack(side=tk.LEFT,
padx=(0, 5))

        self.category_var = tk.StringVar()
        self.category entry =
ttk.Entry(self.category_frame,
textvariable=self.category_var)
        self.category_entry.pack(side=tk.LEFT,
fill=tk.X, expand=True)

        self.predict_button =
ttk.Button(self.category_frame, text="🔍
Predict", command=self._predict_category)

self.predict_button.pack(side=tk.RIGHT,
padx=(5, 0))

self.predict_button.state(['disabled'])

        self.status_frame =
ttk.Frame(self.right_frame)
        self.status_frame.pack(fill=tk.X,
pady=(0, 5))

        self.status_label =
ttk.Label(self.status_frame, text="Status:")
        self.status_label.pack(side=tk.LEFT,
padx=(0, 5))

        self.status_var =
tk.StringVar(value="normal")

        self.normal_radio = ttk.Radiobutton(
            self.status_frame,
            text="Normal",
            variable=self.status_var,
            value="normal"
        )
        self.normal_radio.pack(side=tk.LEFT,
padx=(0, 10))

        self.todo_radio = ttk.Radiobutton(
            self.status_frame,
            text="To-Do",
            variable=self.status_var,
            value="todo"
        )
        self.todo_radio.pack(side=tk.LEFT,
padx=(0, 10))

        self.done_radio = ttk.Radiobutton(
            self.status_frame,
            text="Done",
            variable=self.status_var,
            value="done"
        )
        self.done_radio.pack(side=tk.LEFT)

        self.deadline_frame =
ttk.Frame(self.right_frame)
        self.deadline_frame.pack(fill=tk.X,
pady=(0, 5))

        self.deadline_label =
ttk.Label(self.deadline_frame,
text="Deadline:")
        self.deadline_label.pack(side=tk.LEFT,
padx=(0, 5))

        self.has_deadline_var =
tk.BooleanVar(value=False)
        self.has_deadline_check =
ttk.Checkbutton(
            self.deadline_frame,
            text="Set Deadline",
            variable=self.has_deadline_var,

command=self._toggle_deadline_widgets
        )

self.has_deadline_check.pack(side=tk.LEFT,
padx=(0, 10))

        self.deadline_date_frame =
ttk.Frame(self.deadline_frame)

self.deadline_date_frame.pack(side=tk.LEFT,
padx=(0, 10))

        self.deadline_year_var =
tk.StringVar(value=str(datetime.now().year))
        self.deadline_month_var =
tk.StringVar(value=str(datetime.now().month))
        self.deadline day var =
tk.StringVar(value=str(datetime.now().day))

        self.deadline_year_spin = ttk.Spinbox(
            self.deadline_date_frame,
            from_=datetime.now().year,
            to=datetime.now().year + 10,
            width=6,

textvariable=self.deadline_year_var
        )
        self.deadline_month_spin =
ttk.Spinbox(
            self.deadline_date_frame,
            from_=1,
            to=12,
            width=3,
```

```
textvariable=self.deadline_month_var
        )
        self.deadline_day_spin = ttk.Spinbox(
            self.deadline_date_frame,
            from_=1,
            to=31,
            width=3,
            textvariable=self.deadline_day_var
        )

        ttk.Label(self.deadline_date_frame,
text="Year:").pack(side=tk.LEFT)

self.deadline_year_spin.pack(side=tk.LEFT,
padx=(0, 5))
        ttk.Label(self.deadline_date_frame,
text="Month:").pack(side=tk.LEFT)

self.deadline_month_spin.pack(side=tk.LEFT,
padx=(0, 5))
        ttk.Label(self.deadline_date_frame,
text="Day:").pack(side=tk.LEFT)

self.deadline_day_spin.pack(side=tk.LEFT)

        self.deadline_time_frame =
ttk.Frame(self.deadline_frame)

self.deadline_time_frame.pack(side=tk.LEFT)

        self.deadline_hour_var =
tk.StringVar(value="12")
        self.deadline_minute_var =
tk.StringVar(value="00")

        self.deadline_hour_spin = ttk.Spinbox(
            self.deadline_time_frame,
            from_=0,
            to=23,
            width=3,

textvariable=self.deadline_hour_var
        )
        self.deadline_minute_spin =
ttk.Spinbox(
            self.deadline_time_frame,
            from_=0,
            to=59,
            width=3,

textvariable=self.deadline_minute_var
        )

        ttk.Label(self.deadline_time_frame,
text="Hour:").pack(side=tk.LEFT)

self.deadline_hour_spin.pack(side=tk.LEFT,
padx=(0, 5))
        ttk.Label(self.deadline_time_frame,
text="Min:").pack(side=tk.LEFT)

self.deadline_minute_spin.pack(side=tk.LEFT)

        self._toggle_deadline_widgets()

        self.text_area =
tk.Text(self.right_frame, wrap='word',
font=('Arial', 12))
        self.text_area.pack(fill=tk.BOTH,
expand=True, pady=(0, 5))

        self.button_frame =
ttk.Frame(self.right_frame)
        self.button_frame.pack(fill=tk.X)

        self.save_button =
ttk.Button(self.button_frame, text="Save
Note", command=self._save_current_note)
        self.save_button.pack(side=tk.LEFT,
padx=(0, 5))

        self.completed_date_label =
ttk.Label(self.right_frame, text="")

self.completed_date_label.pack(anchor="e",
pady=(5, 0))

        self.deadline_date_label =
ttk.Label(self.right_frame, text="")

self.deadline_date_label.pack(anchor="e",
pady=(5, 0))

        self.date_label =
ttk.Label(self.right_frame, text="")
        self.date_label.pack(anchor="e",
pady=(5, 0))

        self.analyzer_window =
AnalyzerUI(self.root)

        self._update_connection_status()

    def _toggle_deadline_widgets(self):
        state = '' if
self.has_deadline_var.get() else 'disabled'


self.deadline_year_spin.config(state=state)

self.deadline_month_spin.config(state=state)

self.deadline_day_spin.config(state=state)

self.deadline_hour_spin.config(state=state)

self.deadline_minute_spin.config(state=state)

    def _create_menu(self):
        menu = tk.Menu(self.root)
        self.root.config(menu=menu)

        edit_menu = tk.Menu(menu,
tearoff=False)
        menu.add_cascade(label="Edit",
menu=edit_menu)
        edit_menu.add_command(label="Set as
Normal", command=lambda:
self._set_note_status("normal"))
        edit_menu.add_command(label="Set as
To-Do", command=lambda:
self._set_note_status("todo"))
        edit_menu.add_command(label="Set as
Done", command=lambda:
self._set_note_status("done"))

        ai menu = tk.Menu(menu, tearoff=False)
        menu.add_cascade(label="AI",
menu=ai_menu)

        predict_menu = tk.Menu(ai_menu,
tearoff=False)
        ai menu.add cascade(label="Predict
Category", menu=predict_menu)

predict_menu.add_command(label="Consensus (All
Methods)", command=self._predict_category)

        for i, predictor in
enumerate(self.category_predictor.predictors):
            method_name =
predictor.method_name
            predict_menu.add_command(
                label=method_name,
                command=lambda p=predictor:
self._predict_with_specific_method(p)
            )

        conn_menu = tk.Menu(menu,
```

```
tearoff=False)
        menu.add_cascade(label="Connection",
menu=conn menu)
        conn_menu.add_command(label="Connect
to Server", command=self._reconnect)
        conn_menu.add_command(label="Refresh
Notes from Server",
command=self. request notes from server)

        analytics_menu = tk.Menu(menu,
tearoff=False)
        menu.add_cascade(label="Analytics",
menu=analytics_menu)

analytics_menu.add_command(label="Completion
Speed Analysis",
command=self.analyzer_window.show_completion_s
peed)

analytics_menu.add_command(label="Category
Analysis",
command=self.analyzer_window.show_category_ana
lysis)

analytics_menu.add_command(label="Deadline
Compliance",
command=self.analyzer_window.show_deadline_com
pliance)

analytics_menu.add_command(label="Content
Analysis",
command=self.analyzer_window.show_content_anal
ysis)
        analytics_menu.add_command(label="View
Summary Statistics",
command=self.analyzer_window.show_summary_stat
istics)
        analytics_menu.add_separator()
        analytics_menu.add_command(label="Get
Actionable Insights",
command=self.analyzer_window.show_actionable_i
nsights)

    def _update_notes_listbox(self):
        self.notes_listbox.delete(0, tk.END)
        self.displayed_notes_indices = []

        search_term =
self.search_var.get().lower()

        for i, note in enumerate(self.notes):
            if search_term and not
(search_term in note.title.lower() or

search_term in note.content.lower() or

search_term in note.category.lower()):
                continue


self.displayed_notes_indices.append(i)

            display_text = note.title
            if note.category:
                display_text += f"
[{note.category}]"

            if note.deadline:
                try:
                    deadline_date =
datetime.strptime(note.deadline, "%Y-%m-%d
%H:%M:%S")
                    display_text += f" (Due:
{deadline_date.strftime('%m/%d/%Y')})"

                    if note.is_overdue():
                        display_text += " ⚠"
                except ValueError:
                    pass

            display_index =
len(self.displayed_notes_indices) - 1
            self.notes_listbox.insert(tk.END,
display_text)

            if note.status == "todo":

self.notes_listbox.itemconfig(display_index,
{'bg': self.status_colors["todo"]})
            elif note.status == "done":

self.notes_listbox.itemconfig(display_index,
{'bg': self.status_colors["done"]})

        if self.current_note_index is not
None:
            try:
                displayed_index =
self.displayed_notes_indices.index(self.curren
t_note_index)

self.notes_listbox.selection_clear(0, tk.END)

self.notes_listbox.selection_set(displayed_ind
ex)
            except ValueError:
                if
self.displayed_notes_indices:

self.notes_listbox.selection_set(0)
                    self.current_note_index =
self.displayed_notes_indices[0]
                else:
                    self.current_note_index =
None

    def _filter_notes(self, *args):
        prev_index = self.current_note_index
        self._update_notes_listbox()

        if self.notes_listbox.size() > 0 and
not self.notes_listbox.curselection():

self.notes_listbox.selection_set(0)
            self._on_note_select(None)
        elif self.notes_listbox.size() == 0:
            self.current_note_index = None
            self.title_var.set("")
            self.category_var.set("")
            self.status_var.set("normal")
            self.text_area.delete(1.0, tk.END)
            self.date_label.config(text="")

self.completed_date_label.config(text="")

self.deadline_date_label.config(text="")

    def _on_note_select(self, event):
        if not
self.notes_listbox.curselection():
            return

        listbox_index =
self.notes_listbox.curselection()[0]

        if 0 <= listbox_index <
len(self.displayed_notes_indices):
            actual_index =
self.displayed_notes_indices[listbox_index]
            if 0 <= actual_index <
len(self.notes):
                self.current_note_index =
actual_index
                selected_note =
self.notes[actual_index]


self.title_var.set(selected_note.title)
```

```
self.category_var.set(selected_note.category)

self.status var.set(selected note.status)
                self.text_area.delete(1.0,
tk.END)
                self.text_area.insert(tk.END,
selected_note.content)

                if selected_note.deadline:
                    try:
                        deadline_date =
datetime.strptime(selected_note.deadline, "%Y-
%m-%d %H:%M:%S")

self.has_deadline_var.set(True)

self.deadline_year_var.set(str(deadline_date.y
ear))

self.deadline_month_var.set(str(deadline_date.
month))

self.deadline_day_var.set(str(deadline_date.da
y))

self.deadline_hour_var.set(str(deadline_date.h
our))

self.deadline_minute_var.set(str(deadline_date
.minute))

self._toggle_deadline_widgets()
                    except ValueError:

self.has_deadline_var.set(False)

self._toggle_deadline_widgets()
                else:

self.has_deadline_var.set(False)

self._toggle_deadline_widgets()

                if selected_note.date:

self.date_label.config(text=f"Created:
{selected_note.date}")
                else:

self.date_label.config(text="")

                if selected_note.deadline:

self.deadline_date_label.config(text=f"Deadlin
e: {selected_note.deadline}")
                else:

self.deadline date label.config(text="")

                if selected_note.status ==
"done":

self.completed_date_label.config(text=f"Comple
ted: {selected note.completion date}")
                else:

self.completed_date_label.config(text="")

    def _clear_editor(self):
        self.title_var.set("")
        self.category_var.set("")
        self.status_var.set("normal")
        self.text_area.delete(1.0, tk.END)
        self.date_label.config(text="")

self.deadline_date_label.config(text="")

self.completed_date_label.config(text="")

    def _set_note_status(self, status):
        if self.current_note_index is None:
            messagebox.showinfo("No Note
Selected", "Please select a Note first.")
            return

        self.status_var.set(status)

        self. save current note()
```

predictor_ui.py:

```
import tkinter as tk
from tkinter import messagebox, ttk
import matplotlib

matplotlib.use('TkAgg')
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg

import numpy as np
from ..UI.notepad_interface import
NotepadInterface
from ..Stats.note_analytics import
NotesAnalytics

class PredictorUI (NotepadInterface):
    def _update_predict_button_state(self):
        categorized_notes = [note for note in
self.notes if note.category.strip()]

        if len(categorized_notes) >= 5:

self.predict_button.state(['!disabled'])
        else:

self.predict_button.state(['disabled'])

    def _predict_category(self):
        if self.current_note_index is None:
            messagebox.showinfo("No Note",
"Please select a Note to categorize.")
            return

        categorized_notes = [note for note in
self.notes if note.category.strip()]
        if len(categorized_notes) < 5:
            messagebox.showinfo("Not Enough
Data",
                                "You need at
least 5 categorized notes to use this
feature.")
            return

        current_content =
self.text_area.get(1.0, tk.END).strip()
        if not current_content:
            messagebox.showinfo("Empty Note",
"The Note is empty. Add some content first.")
            return

        results =
self.category_predictor.predict_category(curre
nt_content, categorized_notes)

        consensus_category =
self.category_predictor.get_consensus_predicti
on(results)


self.category_var.set(consensus_category)

        details = "Predictions by method:\n\n"
        for method, result in results.items():
            prediction = result['prediction']
            scores =
result['confidence_scores']

            if scores:
                top_scores =
```

```
sorted(scores.items(), key=lambda x: x[1],
reverse=True)[:3]
                score_text = ",
".join([f"{cat}: {score:.2f}" for cat, score
in top_scores])
                details += f"{method}:
{prediction} ({score_text})\n"
            else:
                details += f"{method}:
{prediction}\n"

        messagebox.showinfo("Category
Prediction",
                            f"Consensus
prediction:
{consensus_category}\n\n{details}")

    def _predict_with_specific_method(self,
predictor):
        if self.current_note_index is None:
            messagebox.showinfo("No Note",
"Please select a Note to categorize.")
            return

        categorized_notes = [note for note in
self.notes if note.category.strip()]
        if len(categorized_notes) < 5:
            messagebox.showinfo("Not Enough
Data",
                                "You need at
least 5 categorized notes to use this
feature.")
            return

        current_content =
self.text_area.get(1.0, tk.END).strip()
        if not current_content:
            messagebox.showinfo("Empty Note",
"The Note is empty. Add some content first.")
            return

        best_category, scores =
predictor.predict_category(current_content,
categorized_notes)

        self.category_var.set(best_category)

        details = ""
        if scores:
            top_scores =
sorted(scores.items(), key=lambda x: x[1],
reverse=True)[:3]
            details = "\n\nTop matches:\n" +
"\n".join([f"{cat}: {score:.2f}" for cat,
score in top_scores])

        messagebox.showinfo(f"Category
Prediction ({predictor.method_name})",
                            f"Predicted
category: {best_category}{details}")

    def _show_completion_forecast_dialog(self,
forecast):
        dialog = tk.Toplevel(self.root)
        dialog.title("Completion Probability
Forecast")
        dialog.geometry("600x500")
        dialog.transient(self.root)
        dialog.grab_set()

        category = forecast["category"]
        probability =
forecast["completion_probability"] * 100
        confidence = forecast["confidence"] *
100

        info_frame = ttk.Frame(dialog,
padding=10)
        info_frame.pack(fill="x", expand=True)

        ttk.Label(info_frame, text=f"Predicted
Category: {category}",
                  font=("TkDefaultFont", 12,
"bold")).pack(anchor="w", pady=5)

        ttk.Label(info_frame,
                  text=f"Probability of On-
Time Completion: {probability:.1f}%",
                  font=("TkDefaultFont",
11)).pack(anchor="w", pady=2)

        ttk.Label(info_frame,
                  text=f"Prediction
Confidence: {confidence:.1f}%",
                  font=("TkDefaultFont",
11)).pack(anchor="w", pady=2)

        if "stats" in forecast and
isinstance(forecast["stats"], dict):
            stats = forecast["stats"]
            if "completed_notes" in stats and
"total_notes" in stats:
                completion_text = (f"Category
History: {stats['completed_notes']} of "

f"{stats['total_notes']} notes completed")
                ttk.Label(info_frame,
text=completion_text).pack(anchor="w", pady=2)

            if "on_time_notes" in stats and
"completed_notes" in stats and
stats["completed_notes"] > 0:
                on_time_text = (f"On-Time
Rate: {stats['on_time_notes']} of "

f"{stats['completed_notes']} completed on
time")
                ttk.Label(info_frame,
text=on_time_text).pack(anchor="w", pady=2)

            if "avg_completion_time" in stats
and stats["avg_completion_time"] is not None:
                time_text = f"Average
Completion Time:
{stats['avg_completion_time']:.1f} hours"
                ttk.Label(info_frame,
text=time_text).pack(anchor="w", pady=2)

        viz_frame = ttk.Frame(dialog,
padding=10)
        viz_frame.pack(fill="both",
expand=True)

        figure = Figure(figsize=(5, 3),
dpi=100)
        ax = figure.add_subplot(111,
projection='polar')

        self._create_gauge_chart(ax,
probability / 100)

        canvas = FigureCanvasTkAgg(figure,
viz_frame)
        canvas.draw()

canvas.get_tk_widget().pack(fill="both",
expand=True)

        interpretation =
self._get_probability_interpretation(probabili
ty / 100)
        ttk.Label(dialog, text=interpretation,
wraplength=550,
                  justify="left",
padding=10).pack(fill="x")

        btn_frame = ttk.Frame(dialog,
padding=10)
        btn_frame.pack(fill="x")
```

```
    def _apply_predicted_category(self,
category, dialog=None):
        if self.current_note_index is not
None:
            self.category_var.set(category)
            if dialog:
                dialog.destroy()

    def _create_gauge_chart(self, ax,
probability):
        ax.clear()

        if probability >= 0.7:
            color = 'green'
        elif probability >= 0.4:
            color = 'orange'
        else:
            color = 'red'

        ax.set_theta_zero_location("N")
        ax.set_theta_direction(-1)
        ax.set_ylim(0, 1)
        ax.set_xlim(-np.pi / 2, np.pi / 2)

        ax.set_frame_on(False)
        ax.set_xticks([])
        ax.set_yticks([])

        theta = np.linspace(-np.pi / 2, np.pi
/ 2, 100)
        radius = 0.8
        ax.plot(theta, [radius] * 100,
'lightgray', linewidth=15)

        value_theta = np.linspace(-np.pi / 2,
-np.pi / 2 + np.pi * probability, 100)
        ax.plot(value_theta, [radius] *
len(value_theta), color, linewidth=15)

        ax.text(0, 0, f"{probability *
100:.0f}%",
                ha='center', va='center',
fontsize=24, fontweight='bold')

        ax.text(-np.pi / 2, radius + 0.1,
"0%", ha='center', va='center')
        ax.text(0, radius + 0.2, "On-Time
Completion Probability", ha='center',
va='center',
                fontsize=12,
fontweight='bold')
        ax.text(np.pi / 2, radius + 0.1,
"100%", ha='center', va='center')

    def _get_probability_interpretation(self,
probability):
        if probability >= 0.8:
            return ("This note has a high
probability of being completed on time. "
                    "Notes in this category
have historically been completed successfully
"
                    "and within deadline.")
        elif probability >= 0.6:
            return ("This note has a good
chance of being completed on time. "
                    "Most notes in this
category have been completed before deadline,
"
                    "but there have been some
delays.")
        elif probability >= 0.4:
            return ("This note has a moderate
chance of being completed on time. "
                    "Notes in this category
have a mixed history of meeting deadlines.")
        elif probability >= 0.2:
            return ("This note has a lower
chance of being completed on time. "
                    "Notes in this category
have frequently missed deadlines or "
                    "remained incomplete.")
        else:
            return ("This note has a very low
probability of being completed on time. "
                    "Notes in this category
have historically struggled with completion "
                    "or meeting deadlines.")

    def _create_ui(self):
        super()._create_ui()

        self.recommend_button = ttk.Button(
            self.button_frame,
            text="Recommend Deadline",
            command=self._recommend_deadline
        )

self.recommend_button.pack(side="left",
padx=5)

    def _recommend_deadline(self):
        if self.current_note_index is None:
            messagebox.showinfo("No Note",
"Please select a Note to analyze.")
            return

        current_content =
self.text_area.get(1.0, tk.END).strip()
        if not current_content:
            messagebox.showinfo("Empty Note",
"The Note is empty. Add some content first.")
            return

        current_category =
self.category_var.get().strip()

        analytics = NotesAnalytics()
        analytics.set_notes(self.notes)

        recommendation =
analytics.get_deadline_recommendation(current_
content,

category=current_category if current_category
else None)

        if "error" in recommendation:

messagebox.showinfo("Recommendation Error",
recommendation["error"])
            return


self._show_deadline_recommendation_dialog(reco
mmendation)

    def
_show_deadline_recommendation_dialog(self,
recommendation):
        dialog = tk.Toplevel(self.root)
        dialog.title("Deadline
Recommendation")
        dialog.geometry("650x600")
        dialog.transient(self.root)
        dialog.grab_set()

        main_frame = ttk.Frame(dialog)
        main_frame.pack(fill="both",
expand=True)

        canvas = tk.Canvas(main_frame)
        scrollbar = ttk.Scrollbar(main_frame,
orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e:
canvas.configure(scrollregion=canvas.bbox("all
```

```
"))
        )

        canvas.create_window((0, 0),
window=scrollable_frame, anchor="nw")

canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both",
expand=True)
        scrollbar.pack(side="right", fill="y")

        content_frame =
ttk.Frame(scrollable_frame, padding=15)
        content_frame.pack(fill="both",
expand=True)

        category = recommendation["category"]
        ttk.Label(content_frame,
                  text=f"Smart Deadline
Recommendation",
                  font=("TkDefaultFont", 14,
"bold")).pack(anchor="w", pady=(0, 5))

        ttk.Label(content_frame,
                  text=f"Category:
{category}",
                  font=("TkDefaultFont",
12)).pack(anchor="w", pady=(0, 10))

        rec_hours =
recommendation["recommended_hours"]
        rec_deadline =
recommendation["recommended_deadline"]
        word_count =
recommendation["word_count"]

        rec_frame = ttk.Frame(content_frame,
padding=5, relief="solid", borderwidth=1)
        rec_frame.pack(fill="x", pady=5)

        deadline_text = f"✓ Recommended
Deadline: {rec_deadline}"
        ttk.Label(rec_frame,
text=deadline_text,
                  font=("TkDefaultFont", 12,
"bold")).pack(anchor="w", pady=5)

        time_text = f"✓ Estimated Time
Required: {rec_hours} hours"
        ttk.Label(rec_frame,
text=time_text).pack(anchor="w", pady=(0, 5))

        ttk.Separator(content_frame,
orient="horizontal").pack(fill="x", pady=10)

        if "estimated_hours" in
recommendation:
            ttk.Label(content_frame,
text="Estimation Methods:",
                      font=("TkDefaultFont",
11, "bold")).pack(anchor="w", pady=(5, 3))

            estimates =
recommendation["estimated_hours"]
            est_frame =
ttk.Frame(content_frame)
            est_frame.pack(fill="x", pady=5)

            row = 0
            if estimates["simple_avg"] is not
None:
                ttk.Label(est_frame,
text="Historical Average:").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
                ttk.Label(est_frame,
text=f"{estimates['simple_avg']:.1f}
hours").grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            if estimates["regression"] is not
None:
                ttk.Label(est_frame,
text="Regression Analysis:").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
                ttk.Label(est_frame,
text=f"{estimates['regression']:.1f}
hours").grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            if estimates["word_based"] is not
None:
                ttk.Label(est_frame,
text="Word Count Based:").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
                ttk.Label(est_frame,
text=f"{estimates['word_based']:.1f}
hours").grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            if estimates["weighted"] is not
None:
                ttk.Label(est_frame,
text="Weighted Estimate:").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
                ttk.Label(est_frame,
text=f"{estimates['weighted']:.1f}
hours").grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            if estimates["time_adjusted"] is
not None:
                ttk.Label(est_frame,
text="Time-of-Day Adjusted:").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
                ttk.Label(est_frame,
text=f"{estimates['time_adjusted']:.1f}
hours").grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            ttk.Label(est_frame, text="Final
(with safety margin):").grid(row=row,
column=0, sticky="w", padx=(15, 10),

pady=2)
            ttk.Label(est_frame,
text=f"{recommendation['recommended_hours']}
hours",
                      font=("TkDefaultFont",
10, "bold")).grid(row=row, column=1,
sticky="w", pady=2)

            row += 1

            if "completion_probability" in
recommendation:
                ttk.Label(est_frame,
text="Completion Probability:").grid(row=row,
column=0, sticky="w", padx=(15, 10),
```

```
pady=2)
                ttk.Label(est frame,
text=f"{recommendation['completion_probability
']}%",
                          font=("TkDefaultFont",
10, "bold")).grid(row=row, column=1,
sticky="w", pady=2)

            if "confidence_intervals" in
recommendation and
recommendation["confidence_intervals"]:
                ttk.Separator(content_frame,
orient="horizontal").pack(fill="x", pady=10)
                ttk.Label(content_frame,
text="Confidence Intervals:",
                          font=("TkDefaultFont",
11, "bold")).pack(anchor="w", pady=(5, 3))

                ci_frame =
ttk.Frame(content_frame)
                ci_frame.pack(fill="x",
pady=5)

                ttk.Label(ci_frame,
text="Confidence Level",
                          font=("TkDefaultFont",
10, "bold")).grid(row=0, column=0, sticky="w",
padx=(15, 10), pady=(0, 5))
                ttk.Label(ci_frame, text="Time
Range",
                          font=("TkDefaultFont",
10, "bold")).grid(row=0, column=1, sticky="w",
padx=(5, 10), pady=(0, 5))
                ttk.Label(ci_frame,
text="Deadline Range",
                          font=("TkDefaultFont",
10, "bold")).grid(row=0, column=2, sticky="w",
padx=(5, 10), pady=(0, 5))

                conf_levels =
sorted(recommendation["confidence_intervals"].
keys(),
                                   key=lambda x:
int(x.replace("%", "")))

                for i, level in
enumerate(conf_levels):
                    interval =
recommendation["confidence_intervals"][level]
                    row_num = i + 1

                    ttk.Label(ci_frame,
text=level).grid(
                          row=row_num, column=0,
sticky="w", padx=(15, 10), pady=2)

                    time range =
f"{interval['lower_hours']} -
{interval['upper_hours']} hours"
                    ttk.Label(ci_frame,
text=time_range).grid(
                          row=row_num, column=1,
sticky="w", padx=(5, 10), pady=2)

                    lower_date =
interval['lower_date'].rsplit(":", 1)[0]
                    upper_date =
interval['upper_date'].rsplit(":", 1)[0]
                    date_range =
f"{lower_date} - \n{upper_date}"
                    ttk.Label(ci_frame,
text=date_range).grid(
                          row=row_num, column=2,
sticky="w", padx=(5, 10), pady=2)

                ci_explanation = (
                    f"These intervals show the
range of time within which your task is likely
to be completed "
                    f"with the corresponding
confidence level. For example, there is a
{conf levels[-1]} chance "
                    f"that you will complete
the task between
{recommendation['confidence_intervals'][conf_l
evels[-1]]['lower_hours']} "
                    f"and
{recommendation['confidence_intervals'][conf_l
evels[-1]]['upper_hours']} hours.")

                ttk.Label(content_frame,
text=ci_explanation, wraplength=580,

justify="left").pack(fill="x", padx=(15, 10),
pady=(5, 0))

            if "explanation" in recommendation
and "probability_explanation" in
recommendation["explanation"]:
                probability_explanation =
recommendation["explanation"]["probability_exp
lanation"]
                ttk.Label(content_frame,
text=probability_explanation, wraplength=580,
                          justify="left",
font=("TkDefaultFont", 10,
"bold")).pack(fill="x", pady=(5, 10))

        if "regression_stats" in
recommendation and
recommendation["regression_stats"]:
            reg_stats =
recommendation["regression_stats"]

            ttk.Separator(content_frame,
orient="horizontal").pack(fill="x", pady=10)
            ttk.Label(content_frame,
text="Regression Analysis Details:",
                      font=("TkDefaultFont",
11, "bold")).pack(anchor="w", pady=(5, 3))

            reg_frame =
ttk.Frame(content_frame)
            reg_frame.pack(fill="x", pady=5)

            if reg_stats["valid"]:
                row = 0

                formula = f"Time =
{reg_stats['slope']:.4f} × Words +
{reg_stats['intercept']:.2f}"
                ttk.Label(reg_frame,
text="Formula:").grid(row=row, column=0,
sticky="w", padx=(15, 10), pady=2)
                ttk.Label(reg_frame,
text=formula).grid(row=row, column=1,
sticky="w", pady=2)
                row += 1

                r_squared =
f"{reg_stats['r_squared']:.2f}
({reg_stats['r_squared'] * 100:.0f}%)"
                ttk.Label(reg frame, text="R-
squared:").grid(row=row, column=0, sticky="w",
padx=(15, 10), pady=2)
                ttk.Label(reg_frame,
text=r_squared).grid(row=row, column=1,
sticky="w", pady=2)
                row += 1

                p_value =
f"{reg_stats['p_value']:.4f}"
                ttk.Label(reg_frame, text="P-
value:").grid(row=row, column=0, sticky="w",
padx=(15, 10), pady=2)
                ttk.Label(reg_frame,
text=p_value).grid(row=row, column=1,
sticky="w", pady=2)
                row += 1
```

```
                std_err =
f"{reg_stats['std err']:.2f} hours"
                ttk.Label(reg_frame,
text="Standard Error:").grid(row=row,
column=0, sticky="w", padx=(15, 10), pady=2)
                ttk.Label(reg_frame,
text=std_err).grid(row=row, column=1,
sticky="w", pady=2)

                interp_text = ""
                if reg_stats["r_squared"] >=
0.7:
                    interp_text = "Strong
correlation between word count and completion
time."
                elif reg_stats["r_squared"] >=
0.4:
                    interp_text = "Moderate
correlation between word count and completion
time."
                else:
                    interp_text = "Weak
correlation between word count and completion
time."

                if reg_stats["p_value"] <
0.05:
                    interp_text += " This
relationship is statistically significant."

                ttk.Label(content_frame,
text=interp_text, wraplength=580,

justify="left").pack(fill="x", padx=(15, 10),
pady=(2, 5))
            else:
                ttk.Label(reg_frame,
text="Insufficient data for reliable
regression analysis.",

wraplength=580).grid(row=0, column=0,
sticky="w", padx=(15, 10), pady=2)

        if "time_of_day_info" in
recommendation and
recommendation["time_of_day_info"]:
            tod_info =
recommendation["time_of_day_info"]

            ttk.Separator(content_frame,
orient="horizontal").pack(fill="x", pady=10)
            ttk.Label(content_frame,
text="Time of Day Impact:",
                      font=("TkDefaultFont",
11, "bold")).pack(anchor="w", pady=(5, 3))

            tod_frame =
ttk.Frame(content_frame)
            tod_frame.pack(fill="x", pady=5)

            row = 0

            if "current_period" in tod_info:
                ttk.Label(tod_frame,
text="Current Period:").grid(row=row,
column=0, sticky="w", padx=(15, 10), pady=2)
                ttk.Label(tod_frame,
text=tod_info["current_period"].capitalize()).
grid(row=row, column=1, sticky="w",

pady=2)
                row += 1

            if "adjustment_factor" in
tod_info:
                factor =
tod_info["adjustment_factor"]
                factor_text = f"{factor:.2f}x"
                if factor > 1.1:
                    factor_text += " (slower
than morning)"
                elif factor < 0.9:
                    factor_text += " (faster
than morning)"
                else:
                    factor_text += " (similar
to morning)"

                ttk.Label(tod_frame,
text="Current Factor:").grid(row=row,
column=0, sticky="w", padx=(15, 10), pady=2)
                ttk.Label(tod_frame,
text=factor_text).grid(row=row, column=1,
sticky="w", pady=2)
                row += 1

            if "efficiency_factors" in
tod_info and tod_info["efficiency_factors"]:
                ttk.Label(tod_frame,
text="Period Comparison:",

font=("TkDefaultFont", 10)).grid(row=row,
column=0, sticky="w", padx=(15, 10), pady=(5,
2))
                row += 1

                factors =
tod_info["efficiency_factors"]
                periods = ["morning",
"afternoon", "evening", "night"]

                for period in periods:
                    if period in factors:
                        ttk.Label(tod_frame,
text=f"{period.capitalize()}:").grid(
                                  row=row, column=0,
sticky="w", padx=(30, 10), pady=1)
                        ttk.Label(tod_frame,
text=f"{factors[period]:.4f}
hours/word").grid(
                                  row=row, column=1,
sticky="w", pady=1)
                        row += 1

        btn_frame = ttk.Frame(dialog)
        btn_frame.pack(fill="x",
side="bottom", pady=10, padx=15)

        explanation = (f"This recommendation
uses multiple prediction methods including "
                       f"regression analysis
of {recommendation.get('notes_analyzed', 0)}
notes "
                       f"in the '{category}'
category. The deadline includes a safety
margin "
                       f"for unexpected
delays.")

        if "time_of_day_info" in
recommendation and
recommendation["time_of_day_info"]:
            explanation += f" Time of day
factors are based on your historical
productivity patterns."

        ttk.Label(content_frame,
text=explanation, wraplength=580,

justify="left").pack(fill="x", pady=10)

        viz_frame = ttk.Frame(content_frame)
        viz_frame.pack(fill="x", pady=10)


self._add_deadline_visualization(viz_frame,
recommendation)

    def _add_deadline_visualization(self,
```

```
frame, recommendation):
        figure = Figure(figsize=(6, 3),
dpi=100)
        ax = figure.add_subplot(111)

        estimates =
recommendation["estimated_hours"]
        methods = []
        values = []

        if estimates["simple_avg"] is not
None:
            methods.append("Historical Avg")

values.append(estimates["simple_avg"])

        if estimates["regression"] is not
None:
            methods.append("Regression")

values.append(estimates["regression"])

        if estimates["word_based"] is not
None:
            methods.append("Word-based")

values.append(estimates["word_based"])

        if estimates["weighted"] is not None:
            methods.append("Weighted")

values.append(estimates["weighted"])

        methods.append("Recommended")

values.append(recommendation["recommended_hour
s"])

        colors = ['lightblue', 'lightblue',
'lightblue', 'lightblue', 'green']

        y_pos = np.arange(len(methods))
        ax.barh(y_pos, values, align='center',
color=colors)
        ax.set yticks(y pos)
        ax.set_yticklabels(methods)
        ax.invert_yaxis()
        ax.set_xlabel('Hours')
        ax.set_title('Deadline Estimation
Methods Comparison')

        for i, v in enumerate(values):
            ax.text(v + 0.1, i, f"{v:.1f}",
va='center')

        canvas = FigureCanvasTkAgg(figure,
frame)
        canvas.draw()

canvas.get_tk_widget().pack(fill="both",
expand=True, pady=10)
```

main.py:

```
import tkinter as tk

from Frontend.UI.noteflow_ui import NoteflowUI

if __name__ == "__main__":
    root = tk.Tk()
    app = NoteflowUI(root)
    root.mainloop()
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

Підтримка особистої ефективності за допомогою застосунка для ведення нотаток з автоматичною категоризацією та аналізом продуктивності

Виконав:
студент 4-го курсу
групи КА-12
Терещук А.В.

Керівник:
Асистент
каф. ММСА
Древаль М.М.

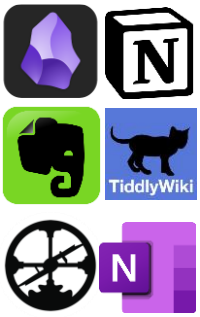
1

Об'єкт дослідження	Предмет дослідження	Метою дослідження
Способи можливого підвищення особистої ефективності за допомогою цифрових інструментів, зокрема автоматизації обробки текстових нотаток, їхньої категоризації та аналізу продуктивності користувача.	Методи для автоматичної обробки та класифікації особистих нотаток та надання аналітики користувача за його ефективність з метою покращення організації часу та продуктивності.	Розробити програмне забезпечення, яке дозволяє користувачам вести структуровані нотатки з автоматичною категоризацією та аналізом продуктивності.

2

Актуальність розглянутої проблеми

У сучасному світі зростає потреба в цифрових інструментах, що допомагають структурувати інформацію, зменшувати когнітивне навантаження й підвищувати особисту ефективність. Історично системи нотування пройшли шлях від воскових табличок і паперових записників до цифрових застосунків із підтримкою штучного інтелекту. Сучасні сервіси на кшталт Notion AI, Mem або Obsidian дозволяють не лише зберігати нотатки, а й аналізувати, класифікувати та прогнозувати дії користувача. Це створює передумови для розробки нових застосунків, що поєднують зручне нотування з елементами аналітики, автоматичного розподілу дедлайнів і персоналізованих рекомендацій на основі історичних даних.



3

Підходи до аналізу текстової інформації

Bag of words з евклідовою, чебишевською і мангеттенською відстанями

$$f_{k,i} = \frac{\sum_{j=1}^{m_k} c_{d_j k,i}}{\sum_{j=1}^{m_k} \sum_{i=1}^{n} c_{d_j k,i}}.$$

Манхеттенська відстань (L_1-норма), формула 2.3.

$$d_M(\vec{a},\vec{b}) = \sum_{i=1}^{n} |a_i - b_i|.$$

Евклідова відстань (L_2-норма), формула 2.4.

$$d_E(\vec{a},\vec{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}.$$

Відстань Чебишева (L_∞-норма), формула 2.5.

$$d_\infty(\vec{a},\vec{b}) = \max_{i=1,\dots,n} |a_i - b_i|.$$

Множинна схожість за індексом Жаккара

$$J(S_{d_{new}}, S_c) = \frac{|S_{d_{new}} \cap S_c|}{|S_{d_{new}} \cup S_c|},$$

Латентно-семантичний аналіз з косинусовою відстанню

$$X_{i,j} \sim TF(t_i, d_j) * IDF(t_i).$$

$$X \approx U_k \Sigma_k V_k^T,$$

$$\vec{v} = \frac{\vec{d}^T \cdot v_k}{\Sigma_k}.$$

$$\vec{C}_c = \frac{\sum_{d_j \in D_c} U_k[:k]}{|D_c|},$$

TF-IDF з косинусовою відстанню

$$TF - IDF(t,d) = TF(t,d) * IDF(t).$$

$$TF(t,d) = \frac{f(t,d)}{|d|}, \quad IDF(t) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right).$$

$$similarity(d_{new}, c) = \frac{\vec{v}_{d_{new}} \cdot \vec{v}_c}{||\vec{v}_{d_{new}}|| * ||\vec{v}_c||},$$

Наївний байєсівський класифікатор

$$P(w_i|C = c_j) = \frac{count(w_i, c_j)}{count(c_j)},$$

$$P(C = c_j|d_{new}) \sim P(C = c_j) \prod_{i=1}^{n} P(w_i|C = c_j).$$

$$\log\left(P(C = c_j|d_{new})\right) \sim \log\left(P(C = c_j)\right) + \sum_{i=1}^{n} \log\left(P(w_i|C = c_j)\right).$$

4

Способи створення аналітики і передбачення дати на основі минулих періодів

Зведена статистика активності користувача за категоріями

$$\mu_t = \frac{1}{|T_c|} \sum_{t \in T_c} t. \qquad median_t = med(T_c).$$

$$\dot{\mu}_t = \sqrt{\frac{1}{|T_c|} \sum_{t \in T_c} (t - \mu_t)^2}. \qquad on\_time\_rate_c = \frac{|\{a \in A_c : a \geq 0\}|}{|A_c|}.$$

Аналіз швидкості виконання нотаток

$$r = \frac{\sum (w_i - \bar{w})(t_i - \bar{t})}{\sqrt{\sum (w_i - \bar{w})^2} \sqrt{\sum (t_i - \bar{t})^2}}, \qquad \hat{t}_i = \beta_0 + \beta_1 w_i,$$

$$\mu_c = E\left[\frac{t_i}{w_i}\right], \ \forall i : c_i = c,$$

$$\mu_h = E\left[\frac{t_i}{w_i}\right], \ \forall i : h_i = h.$$

Аналіз дотримання дедлайнів

$$R_{ot} = \frac{|N_{OT}|}{|N_D|}. \qquad \bar{a} = \frac{\sum_{n \in N_C} a_n}{|N_C|}.$$

$$r_{pb} = \frac{\bar{b}_1 - \bar{b}_0}{s_b} \sqrt{\frac{n_1 n_0}{n(n-1)}}, \qquad P(y_n = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 b_n)}},$$

Рекомендації щодо дедлайну

$$\bar{t} = \frac{1}{n} \sum_{i=1}^{n} t_i. \quad \bar{t}_{per\ word} = \frac{\sum t_i}{\sum w_i}. \quad t_{words} = W \cdot \bar{t}_{per\ word}.$$

$$t_{\text{weighted}} = \frac{\sum w_k \cdot t_k}{\sum w_k}, \quad t_{\text{adj}} = t_{\text{weighted}} \cdot \bar{e}_{\text{current}}.$$

$$t_{rec} = t_{adj} + z \cdot s,$$

$$[t_{\text{adj}} - t_{n-1} \cdot s\sqrt{1 + \frac{1}{n}};\ t_{\text{adj}} + t_{n-1} \cdot s\sqrt{1 + \frac{1}{n}}],$$

5

Архітектурні принципи побудови програми

Реалізація збереження даних

Локальна база даних SQLite організована у вигляді таблиці нотаток, що дозволяє зберігати текстові поля та дати в зручному і надійному форматі. Завдяки транзакційній моделі ACID забезпечується цілісність даних навіть при збоях, а зберігання у вигляді одного файлу спрощує розгортання, резервне копіювання та переносимість.

Реалізація сервера

Бекенд реалізований на C++, що дає змогу працювати з великим обсягом операцій читання та запису, використовуючи підготовлені SQL-запити. Контроль пам'яті через RAII і оптимізації коду забезпечують стабільність і високу продуктивність навіть на слабкому обладнанні.

Реалізація клієнта

Фронтенд створений на Python з Tkinter, що дозволяє розробляти графічний інтерфейс без додаткових залежностей. Такий інтерфейс підтримує кросс-платформеність і легко інтегрується з бібліотеками для аналітики, забезпечуючи доступ до нотаток, редагування і візуалізацію даних.

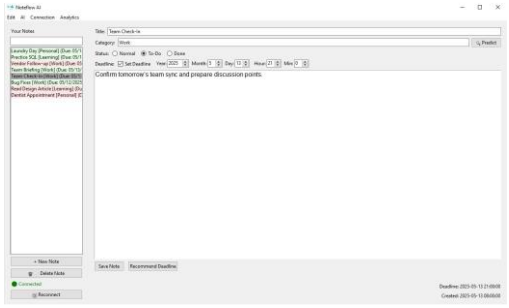
6

Результати роботи серверу

```
andr1ano@DESKTOP-UBJJSRU:/mnt/c/Users/andr1ano/Documents/KPI/ДИПЛОМ/NoteflowAI/Backend$ ./socket_handler
Starting Notes Server...
Database: note_db/notes.db
Port: 8080
Database initialized successfully
Socket server set up on port 8080
Waiting for connections...
New client connected
Received operation: list
Received operation: save
^C
Received signal 2. Shutting down server...
andr1ano@DESKTOP-UBJJSRU:/mnt/c/Users/andr1ano/Documents/KPI/ДИПЛОМ/NoteflowAI/Backend$
```

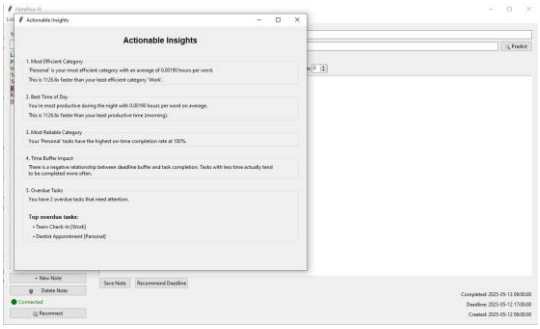
7

Результати роботи клієнта

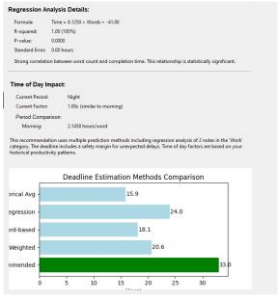


8

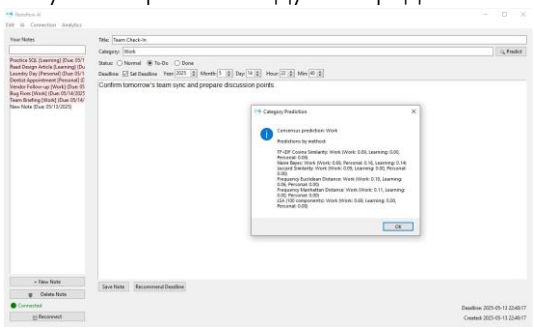
Результати роботи модулю аналітики



Результати роботи модулю аналітики



Результати роботи модулю передбачення



Перспективи розвитку

- Розширення аналітичного функціоналу – впровадження нових методів аналізу продуктивності, зокрема часових рядів і поведінкових моделей, для точнішого виявлення тенденцій та формування індивідуальних порад.
- Покращення інтерфейсу користувача – оптимізація UX/UI з акцентом на персоналізацію та інтерактивні візуалізації для зручнішого сприйняття даних.
- Удосконалення алгоритмів категоризації – використання глибших моделей обробки природної мови для точнішого розуміння змісту нотаток і контекстного поділу за тематиками чи пріоритетами.

ВИСНОВКИ

У результаті дослідження було створено програму для автоматизованого управління текстовими нотатками, яка надає користувачеві інтерфейс для отримання певних аналітичних даних. Було реалізовано засоби автоматичної класифікації нових нотаток за тематикою і вже існуючими категоріями, а також модулі аналітики для представлення деякої статистики по роботі користувача з нотатками. Архітектура системи побудована з урахуванням розділення на клієнтську та серверну частини, що в результаті створює єдину сутність застосунку на базі двох процесів. Імплементована можливість створювати, редагувати та переглядати нотатки, визначати для цих же нотаток автоматично категорії і отримувати деякі інформативні показники щодо виконанням за часом доби, вплив буферу часу на виконання в встановелний строк, рекомендації щодо встановлення строку для нової нотатки тощо.

Noteflow AI
TO MAKE ROUTINE UNIQUE.

ДЯКУЮ ЗА УВАГУ!