Міністерство освіти та науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«ПЕРЕВІРКА СТАТИСТИЧНИХ ГІПОТЕЗ ЯК ОСНОВА А/В ТЕСТУВАННЯ»**

Виконала: студентка 4-го року навчання,

Спеціальності

121 Інженерія програмного забезпечення

Варещук Марія Олегівна

Керівник Крюкова Г. В._____

асистент

Рецензент _____

Кваліфікаційна робота захищена

з оцінкою _____

Секретар ЕК_____

«____» _____ 20____р.

Київ – 2023

ЗМІСТ

АНОТАЦІЯ **3**

ВСТУП **4**

1 ТЕОРЕТИЧНІ ОСНОВИ **6**

1.1 Огляд ключових понять *6*

1.2. Fixed-based horizon tests *8*

1.3 Frequentist interference *10*

1.4 Simple sequential A/B testing *11*

1.5 Bayesian testing *13*

Висновки до розділу 1 ..... *15*

2 РОЗРОБКА ТА АНАЛІЗ МЕТОДІВ ПЕРЕВІРКИ СТАТИСТИЧНИХ ГІПОТЕЗ **17**

2.1. Розробка автоматичних розрахунків за допомогою Python *17*

2.1.1. Автоматизація fixed-based horizon 17

2.1.2. Автоматизація frequentist interference 20

2.1.3. Автоматизація sequential sampling 21

2.1.4. Автоматизація Bayesian testing 22

2.2. Порівняльний аналіз *24*

Висновки до розділу 2 ..... *26*

3 РОЗРОБКА ДОДАТКА **27**

3.1. Огляд існуючих рішень *27*

3.2. Вибір технологій та інструментів розробки *28*

3.3. Розробка архітектури та інтерфейсу програмного додатка *31*

Висновки до розділу 3 ..... *49*

ВИСНОВКИ **50**

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ **52**

ДОДАТКИ **54**

АНОТАЦІЯ

Робота присвячена дослідженню методів A/B тестування для перевірки статистичних гіпотез щодо поведінки користувачів різних додатків та сайтів. Було визначено найбільш ефективні методи для різноманітних бізнес-випадків та створено систему автоматичного розрахунку та аналізу тестів. Система реалізована у вигляді веб-застосунку використовуючи мови програмування Python та JavaScript.

Ключові слова: A/B ТЕСТУВАННЯ, ПЕРЕВІРКА СТАТИСТИЧНИХ ГІПОТЕЗ, БАЄСОВСЬКЕ ТЕСТУВАННЯ, ЧАСТОТНЕ ТЕСТУВАННЯ, АВТОАНАЛІЗ ПРОДУКТОВИХ ТЕСТІВ, МОВА ПРОГРАМУВАННЯ PYTHON, МОВА ПРОГРАМУВАННЯ JAVASCRIPT, ФРЕЙМВОРК FLASK, БІБЛІОТЕКА REACT.

ВСТУП

У швидкозмінному світі технологій та цифрового маркетингу, бізнеси стараються оптимізувати свої онлайн-платформи, щоб досягти максимального залучення користувачів. Для цього вони використовують дані про дії користувачів та аналіз цих даних. За допомогою A/B тестування вони можуть оцінювати та порівнювати різні ітерації веб-сайту або додатку, що дозволяє приймати “data-driven” рішення. Однак ручний запуск та аналіз тестів є ресурсозатратним та схильним до людських помилок процесом.

У цій дипломній роботі розглядається ідея автоматизації в A/B-тестуванні і те, як її можна використовувати для прискорення експериментів. Крім того, досліджено різноманітні способи перевірки статистичних гіпотез та визначено найбільш ефективні рішення для того чи іншого випадку у бізнесі.

У першому розділі будуть розглянуті теоретичні основи A/B тестування. Для цього будуть досліджені та проаналізовані методи перевірки статистичних гіпотез, які дозволяють визначити чи зміни на платформі покращують чи погіршують цільову метрику.

У другому розділі будуть розроблені алгоритми розрахунку та аналізу для кожного з методів A/B тестування. Буде проведений порівняльний аналіз, що дозволить визначити найбільш ефективні методи A/B тестування для конкретних бізнес-випадків.

У третьому розділі буде описано розробку веб-застосунку для автоматичного розрахунку та аналізу тестів. Будуть розглянуті функціональні вимоги, технології, використані для реалізації та інтерфейс застосунку. Результатом буде розробка системи для автоматичного розрахунку та аналізу різних методів A/B тестування.

Актуальність дослідження полягає у тому, що кожен data-driven бізнес приймає рішення на основі результатів A/B тестування. Відповідно швидкий і коректний розрахунок метрик, необхідних для проведення тесту та аналіз результатів є критично важливою задачею.

Областю дослідження є застосування різних методів перевірки статистичних гіпотез для проведення A/B тестування для порівняння різноманітних версій веб-сайту або додатку та визначення версії, що найкраще впливає на цільову метрику бізнесу та на залученість користувачів в цілому.

Предметом дослідження є розробка та аналіз методів перевірки статистичних гіпотез для A/B тестування на веб чи мобільних додатках та створення застосунку, який буде використовувати дані методи для автоматичного розрахунку та аналізу тестів.

Метою дослідження є розробка веб-застосунку для швидкого та коректного аналізу різноманітних методів A/B тестування. Для досягнення цієї мети потрібно проаналізувати різні методи перевірки статистичних гіпотез, зробити їх порівняльний аналіз для визначення найбільш ефективних в тих чи інших випадках в бізнесі та розробити додаток, який зможе автоматично розраховувати та аналізувати результати тестів.

Практичне значення дослідження є у створенні зручного інструменту для ефективного проведення A/B тестування на веб чи мобільних додатках. Створений інструмент може пришвидшити прийняття data-driven рішень у бізнесі та запобігти наявності людських помилок у цьому процесі.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Огляд ключових понять

A/B тестування – це порівняння двох або більше версій чогось (зазвичай веб-сторінка, елемент застосунку, ціна підписки), щоб визначити, яка версія покращує цільові бізнес-показники. Математична статистика є інструментом для проведення порівняння, зокрема вона дозволяє зрозуміти скільки користувачів мають взяти участь в експерименті та чи можемо ми стверджувати про перемогу певної групи базуючись на отриманому результаті. Зазвичай тестують версію, яка не була піддана будь-яким змінам (також називається *контрольною групою*, або *група А*) проти версії з зміненим елементом (*тестова група*, або *група В*). Зміненим елементом може бути дизайн кнопки, ціна підписки, порядок сторінок у застосунку, розташування елементів на сторінці, тощо. Схема A/B тесту:

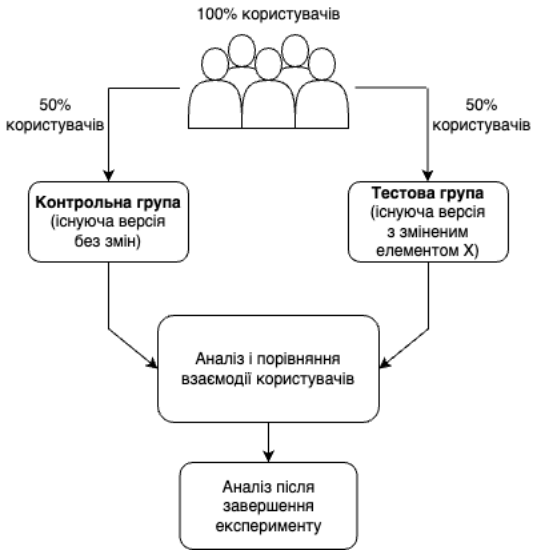


Рисунок 1 - Схема простого A/B тесту

Перед початком тесту необхідно сформулювати продуктову гіпотезу. Наприклад, якщо тестується новий дизайн сторінки оплати, гіпотеза може звучати, як: при зміні дизайну сторінки оплати користувач буде краще розуміти за що і скільки він платить і, відповідно, збільшиться конверсія користувачів з сторінки оплати до успішного оформлення підписки. Саме завдяки правильно

сформульованій продуктовій гіпотезі можна визначити коректну цільову метрику експерименту.

Цільова метрика експерименту – це показник, зміна якого покаже чи є нова версія більш успішною, ніж попередня. Це може бути як і кількісний показник (середня довжина сесії користувача на платформі), так і якісні (відношення кількості користувачів, які зробили оформили підписку до всіх людей у вибірці). Цільова метрика визначається в залежності від бізнес-стратегії та елементу, на який тестується. Наприклад, якщо тестова група бачить інший колір кнопки на сторінці А і гіпотеза звучить як “зміна кольору кнопки на сторінці А збільшить прохідність користувачів до сторінки Б”, то цільовою метрикою може бути відношення кількості користувачів, які потрапили на сторінку Б до кількості людей на сторінці А.

Відношення людей, які зробили певну дію до всіх людей у вибірці також будемо називати *conversion rate (CR).* Математично можемо сказати, що CR – це математичне очікування розподілу Бернуллі.

Мінімальний бажаний результат (Minimal Desirable Effect, MDE) – відсоток зміни цільової метрики експерименту, при досяжності якого ми вважаємо зміни на додатку значущими. Цей показник рахується з мінливості самої метрики та загальної стратегії бізнесу.

Нульова гіпотеза (H0) – гіпотеза про те, що у всьому винен випадок.

Альтернативна гіпотеза (H1) – протилежне до нульової гіпотези (те, що ми стараємось довести).

Якщо нульова гіпотеза вірна, то ймовірність її відхилення є ймовірністю помилки першого роду.

Якщо альтернатива вірна, ймовірність прийняття нульової гіпотези є ймовірністю помилки другого роду.

Рівень значущості (significance level) - представлений як α(альфа), встановлює ймовірність того, що нульова гіпотеза (H0), яка є насправді вірною, буде відхилена. Зазвичай використовується рівень значущості 0,05 (або 5%). Це

означає, що нульова гіпотеза відхиляється, якщо p-значення менше або дорівнює α.

P-значення (p-value) - p-значення показує ймовірність отримання спостережуваних даних або більш екстремальних даних за умови правильності нульової гіпотези. Ймовірність відхилення нульової гіпотези зростає зі зменшенням p-значення. Для оцінки статистичної значущості даних p-значення зазвичай порівнюють з рівнем значущості (α). Результати вважаються статистично значущими, а нульова гіпотеза відхиляється, якщо p-значення менше або дорівнює α.

Статистична потужність (statistical power) визначає, наскільки ймовірно виявити реальну різницю між групами, яка є статистично значущою. Висока потужність тесту вказує на низьку ймовірність помилки другого роду, тобто прийняття нульової гіпотези, коли вона є неправильною. На потужність тесту впливають такі змінні, як розмір вибірки, рівень значущості та волатильність даних.

Довірчий інтервал (confidence interval) - показує діапазон значень на основі вибіркових даних, в якому, як очікується, може перебувати параметр популяції. Найчастіше використовуються довірчий інтервал 95%. Зі збільшенням довірчого інтервалу зменшується точність оцінки, але збільшується впевненість в його охопленні.

1.2. Fixed-based horizon tests

Одним з найбільш популярним методом тестування є fixed-based horizon T-test. Його головна ідея полягає у попередньому розрахунку необхідної кількості користувачів та, відповідно, тривалості тесту. Оцінювати тест можемо лише тоді, коли необхідний розмір вибірки досягнутий. Загальний алгоритм проведення fixed-based horizon тесту:

1. Обрахувати розмір вибірки
2. Запустити тест

3. Очікувати розмір вибірки

4. Оцінити тест

5. Експеримент закінчено

Визначення розміру вибірки має критично важливе значення для отримання статистично значимих результатів. Формула для розрахунку розміру вибірки для fixed-based horizon T-тестів:

$$n = (Z\alpha/2 + Z\beta)^2 * (p1 * (1 - p1) + p2 * (1 - p2)) / (p2 - p1)^2$$

У цій формулі:

- n - необхідний розмір вибірки для кожної з груп;
- $Z\alpha/2$ - критичне значення стандартного нормального розподілу на рівні $\alpha/2$ (рівень значущості, поділений на 2);
- $Z\beta$ - критичне значення стандартного нормального розподілу при β (бажана статистична потужність);
- p1; p2 – базова конверсія для груп A і B відповідно.

Розрахунок приблизної тривалості тесту базується на середній кількості маркетингових витрат та ціни 1 відвідування користувача.

Даний підхід має декілька переваг:

1. Контрольований розподіл ресурсів. Через чітке розуміння вартості тесту компанії мають можливість заздалегідь знати та розподіляти кошти.
2. Передбачувані часові рамки. Використовуючи fixed-based horizon підхід є можливість наперед планувати тести через розуміти тривалості кожного з них.

Проте, існують і недоліки:

1. Складність визначити базові конверсії на малих об’ємах даних. При високій волатильності цільової метрики визначення базової конверсії може бути важкою для реалізації задачею.
2. Негнучкість. Тести з фіксованим горизонтом не адаптуються до різких змін коефіцієнтів конверсій або інших непередбачуваних факторів, що може призвести до неоптимальної витрати ресурсів.

1.3 Frequentist interference

Frequentist interference є більш динамічним методом тестування в порівнянні з fixed-based horizon. Основна ідея полягає у динамічному контролі значущості та потужності тесту. Загальний алгоритм проведення тесту за методом frequentist interference:

1. Запустити тест
2. Оцінити тест

Щоб оцінити результати тестування і варто обчислити статистичні метрики на основі тесту. Для Frequentist interference тестування застосовуються такі статистики:

- Z-статистика: залежно від розміру вибірки та вимог до нормальності даних, на основі t-розподілу Стьюдента або нормального розподілу. Розрахунок статистичної значущості різниці здійснюється шляхом порівняння середніх значень двох вибірок з урахуванням розміру вибірки та дисперсії. Z-статистика розраховується за допомогою наступної формули:

$$Z = (\bar{X}_1 - \bar{X}_2) / \sqrt{[(s_1^2 / n_1) + (s_2^2 / n_2)]},$$

де $\bar{X}_1$ і $\bar{X}_2$ - це відповідні середні першої та другої вибірок;

s_1 і s_2 - це відповідні стандартні відхилення вибірок;

n_1 і n_2 - це обсяги вибірок.

- P-значення: Якщо нульова гіпотеза про відсутність різниці між групами вірна, то це ймовірність отримання різниці між вибірковими середніми, яка дорівнює або перевищує фактичну різницю. Існує більше доказів проти нульової гіпотези, отже, чим менше P-значення, тим краще.
- Довірчий інтервал: З певним рівнем довіри цей інтервал містить діапазон ймовірних значень різниці між середніми значеннями груп. Наприклад, довірчий інтервал з рівнем довіри 95% показує, що різниця між середніми значеннями груп знаходиться всередині цього інтервалу з довірчою

ймовірністю 95%. Для обчислення довірчого інтервалу можна використовувати Z-статистику.

3. Якщо критерій зупинки досягнуто - експеримент закінчено. В іншому випадку, потрібно продовжувати експеримент

1.4 Simple sequential A/B testing

Sequential sampling усуває проблему "підглядання", спричинену ентузіазмом експериментаторів, які використовують традиційні методи фіксованої вибірки, дозволяючи експериментатору достроково завершити випробування, якщо тестова група виявиться переможцем.

Загальний алгоритм тесту:

1. На початку експерименту фіксуємо розмір вибірки N та розподіл вибірок 50/50
2. Позначимо за T кількість успіхів у тестовій групі
3. Позначимо за C кількість успіхів у контрольній групі
4. якщо $T - C >= 2\sqrt{N}$, зупинити тест. Тестова група – переможець
5. якщо $T + C >= N$, зупинити тест. Переможця немає

У деяких випадках даний метод може скоротити кількість спостережень, необхідних для успішного експерименту, на 50% або більше. При малих параметрах Бернуллі або низьких коефіцієнтах перетворення цей метод працює дуже добре. Він працює менш ефективно, коли коефіцієнт перетворення великий, хоча за таких обставин класичні підходи з фіксованою вибіркою повинні працювати так само добре.

За основу цього методу було взято відому Gambler’s Ruin задачу. Розглянемо гравця, який кидає монету проти суперника з d жетонами. Опонент дасть гравцеві жетон, якщо той правильно передбачить орел або решку; в іншому випадку він втратить жетон. Гра закінчується, коли у гравця (або суперника) закінчуються жетони.

Спочатку припустимо, що кидання монети є чесним. Ймовірність того, що гравець, який починає з d жетонів, збанкрутує після рівно n підкидань монети, забезпечується за умови, що опонент має необмежену кількість жетонів для ставок [1]:

$$r_{n,d} = \frac{d}{n}\binom{n}{(n+d)/2}2^{-n}.$$

Сума по $r_{n,d}$ дає ймовірність того, що гравець програє все за перші N підкидань монети:

$$R_{N,d} = \sum_{n=1}^{N} r_{n,d} = \sum_{n=1}^{N} \frac{d}{n}\binom{n}{(n+d)/2}2^{-n}$$

Який зв'язок між поганим гравцем і A/B тестуванням? Величина T-C з загального алгоритму збільшується на одиницю з кожним успіхом в основній групі і зменшується на одиницю з кожним успіхом в контрольній групі. Нехай d=T-C. Тепер, починаючи з d=0, ймовірність того, що d зросте до деякого значення d*, дорівнює ймовірності того, що у гравця, який починає з d* фішок, заберуть всі його фішки. Таким чином, односторонній послідовний тест на статистичну значущість для A/B тесту може бути створений за допомогою наступного рівняння, яке представляє кумулятивний шанс розорення з плином часу. Необхідно вибрати найменше додатне значення d*, яке задовольняє наступним умовам для заданого N та рівня значущості α:

$$\sum_{n=1}^{N} \frac{d^*}{n}\binom{n}{(n+d^*)/2}2^{-n} < \alpha$$

Нульова гіпотеза про те, що тестова група не є кращою за контрольну відхиляється, якщо значення d досягає критичного значення d* за N або менше успіхів (підкидань монети). Якщо критичного значення не досягнуто, нульова гіпотеза приймається.

Критичні значення d* можна точно передбачити за допомогою [2]:

$d* \approx z_{\alpha/2}\sqrt{N},$

де $z_{\alpha/2}$ - квантиль нормального розподілу. Для α=0.05, $z_{\alpha/2}$ =1.96. Емпіричним правилом є $d^* \approx 2\sqrt{N}$ при округленні z до 2.

Процес можна завершити достроково у тому сенсі, що тест буде завершено, як тільки буде досягнуто d*. Однак його не слід припиняти до того, як буде досягнуто d* і зафіксовано N успіхів. Аналогічно, продовження тесту після N успіхів зробить статистичні висновки недійсними. Цей підхід може бути використаний для скорочення кількості спостережень, необхідних для завершення тесту.

1.5 Bayesian testing

Байєсівське тестування - це ефективний інструмент, який пропонує ймовірнісну основу для оцінки ефективності тестової групи та прийняття рішень на основі даних. Принципи байєсівського тестування: Байєсівська статистика, яка включає аналіз апріорного розподілу, є основою байєсівського тестування. Байєсівське тестування дозволяє інтегрувати попередню інформацію, на відміну від стандартних частотних методів, які використовують лише спостережувані дані, що призводить до більш складного та обґрунтованого прийняття рішень.

Оцінка апостеріорного розподілу параметрів на основі спостережуваних даних є основною метою байєсівського тестування. Після врахування як минулих знань, так і даних, зібраних під час експерименту, цей розподіл дає оновлені знання щодо параметрів, які нас цікавлять.

Ключові поняття в байєсівському аналізі:

Ще до того, як були зібрані дані, перші припущення щодо параметрів потрібно відобразити в апріорному розподілі. Він включає в себе попередні дослідження (data-driven підхід) або професійні судження (суб’єктивні дані).

Функція правдоподібності показує ймовірність того, що дані відповідають певному набору значень параметрів. Вона вимірює, наскільки добре дані відповідають різним налаштуванням параметрів. Завдяки теоремі

Байєса, яка оновлює попередні переконання до апостеріорного розподілу, функція правдоподібності є важливою частиною байєсівського тестування.

Розподіл параметрів, який було оновлено з урахуванням спостережуваних даних, відомий як апостеріорний розподіл. Він створюється шляхом нормалізації попереднього розподілу і множення його на функцію правдоподібності. Апостеріорний розподіл забезпечує ретельне представлення невизначеності параметрів і дозволяє робити висновки та приймати рішення на основі найновіших даних.

Складні апостеріорні розподіли, які не можуть бути визначені аналітично, часто використовуються в байєсівському тестуванні. Зазвичай апостеріорний розподіл апроксимується за допомогою методів ланцюгових марковських моделей Монте-Карло (MCMC). Для ефективної вибірки та висновків методи MCMC створюють серію значень параметрів, які збігаються до бажаного апостеріорного розподілу.

Припустимо, що $P(\lambda) = f(\lambda; a, b)$. Припустимо, що є n осіб і що c з них здійснили цільову дію. Враховуючи, що $f(x; a, b)$ – Бета розподіл, тоді апостеріорна ймовірність дорівнює [3]:

$$P(\lambda|n, c) = f(x; a + c, b + (n - c))$$

Відомо з бета розподілу, що:

$$\mu = \frac{a}{a+b} \; \sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$$

Звідси:

$b = a(1 - \mu)/\mu.$

Тоді:

$$\sigma^2 = \frac{a^2(1-\mu)}{\mu(a+b)^2(a+b+1)} = \frac{1-\mu}{\mu(1+(1-\mu)/\mu^2)^2(a/\mu+1)}.$$

Звідси зв’язок медіани та дисперсії із параметрами Бета розподілу:

$$a = \frac{1-\mu-\sigma^2\mu^3}{\sigma^2\mu^2}; \; b = \frac{(1-\mu)[1-\mu-\sigma^2\mu^3]}{\sigma^2\mu^3}$$

Припустимо, що ми оцінюємо варіанти A і B для nA і nB осіб відповідно. Тоді апостеріорна ймовірність для кожного з $PA(\lambda A)$ $PB(\lambda B)$ може бути визначена наступним чином:

$P(\lambda A, \lambda B) = PA(\lambda A)PB(\lambda B)$

Ймовірність помилки:

$$E[\mathcal{I}](A) = \int_0^1 \int_0^{\lambda_A} P(\lambda_A, \lambda_B)\, d\lambda_B\, d\lambda_A;$$

$$E[\mathcal{I}](B) = \int_0^1 \int_{\lambda_A}^1 P(\lambda_A, \lambda_B)\, d\lambda_B\, d\lambda_A$$

Функція втрат, яка приймає певні значення λA λB, представляє кількість підйому (lift), яку можна передбачити, що буде втрачено при виборі запропонованого варіанту:

$\mathscr{L}(\lambda A, \lambda B, A) = \max(\lambda B - \lambda A, 0)$;

$\mathscr{L}(\lambda A, \lambda B, B) = \max(\lambda A - \lambda B, 0)$.

Очікувані витрати при об’єднаній апостеріорній ймовірності, де ? – це значення A чи B:

$$E[\mathcal{L}_{(?)}] = \int_0^1 \int_0^1 \mathcal{L}(\lambda_A, \lambda_B, ?) P(\lambda_A, \lambda_B)\, d\lambda_B\, d\lambda_A.$$

Висновки до розділу 1

У цьому теоретичному огляді було розглянути кілька підходів, які часто використовуються в A/B-тестуванні, зокрема fixed-based horizon, frequentist interference, sequential sampling та Bayesian тестування.

Кожен з цих підходів має свої переваги та недоліки. Метод з фіксованим горизонтом, хоча йому може бракувати гнучкості, пропонує передбачуваність і планування. Frequentist interference забезпечує адаптивне прийняття рішень, але критерії зупинки повинні бути ретельно продумані. Методи частотного висновку, хоча дозволяють більш динамічний розрахунок, можуть працювати лише для пропорційних тестів. Використання апріорного розподілу в

Байєсівському тестуванні, хоча і дозволяє більш глибокий аналіз даних для тесту, просте не завжди є очевидним розрахунком.

2 РОЗРОБКА ТА АНАЛІЗ МЕТОДІВ ПЕРЕВІРКИ СТАТИСТИЧНИХ ГІПОТЕЗ

2.1. Розробка автоматичних розрахунків за допомогою Python

У цьому розділі будуть розроблені алгоритми для проведення A/B тестування, включаючи кроки планування тесту, обробку результатів та статистичний аналіз. Розробка буде відбуватись за допомогою мови програмування Python, зокрема бібліотек numpy та scipy. Кожен крок буде розглянутий детально з поясненням необхідних методів та процедур. Методи тестування, що будуть розглянуті:

- Fixed-based horizon
- Frequentist interference
- Sequential sampling
- Bayesian testing

2.1.1. Автоматизація fixed-based horizon

Для початку, опишемо псевдокодом загальний алгоритм проведення тесту:

```
if Nreal>Sample size then
      if pvalue < α then
            if Lift > 0 then
                  verdict = «B»
            else
                  verdict = «A»
            end if
      else
            verdict = «no difference»
      end if
else
      verdict = «None»
end if
```

Отже, почнемо з розрахунку Sample size базуючись на базовій конверсії, мінімального бажаного ефекту, статистичній потужності та рівню значущості:

Формула розрахунку обсягу вибірки оцінює спільну варіабельність між контрольною та тествою групами шляхом обчислення об'єднаного стандартного відхилення (pool_sd). Вона враховує коефіцієнти конверсії обох груп (p1 і p2), а також їхні відповідні додаткові ймовірності (1 - p1 і 1 - p2).

Ми робимо припущення, що генеральна дисперсія обох груп є еквівалентною для обчислення об'єднаного стандартного відхилення. Використовуючи це припущення, ми можемо обчислити загальне стандартне відхилення, яке відображає варіабельність сукупності.

Об'єднане стандартне відхилення обчислюється наступним чином:

```
pool_sd = np.sqrt((p1 * (1 - p1) + p2 * (1 - p2)) / 2)
```

Використовуючи відповідні ймовірності як ваги, дисперсії контрольної та варіаційної груп об'єднуються в цьому обчисленні. Для обчислення стандартного відхилення використовуємо квадратний корінь.

Щоб врахувати варіабельність популяції та розрахувати необхідний обсяг вибірки для знаходження відповідного мінімального виявленого ефекту (MDE) з прийнятною статистичною потужністю та рівнем значущості, використовуємо об'єднане стандартне відхилення.

Для визначення обсягу вибірки використовується формула:

```
z_score = stats.norm.ppf(1 - alpha / 2)
sample_size = np.ceil((2 * (z_score + stats.norm.ppf(power))**2) / mde**2 *
pool_sd**2)
```

У цій формулі об'єднане стандартне відхилення (pool_sd) множиться на квадратний корінь найменшого помітного ефекту (MDE). Ця формула слугує заміною для частини дисперсії при розрахунку розміру вибірки і допомагає визначити рівень точності, необхідний для визначення цільового розміру ефекту.

На цьому етапі ми запускаємо тест, очікуючи необхідний розмір вибірки на кожній з варіацій. Коли розмір вибірки досягнутий – розраховуємо результати.

Зібрані дані з кожної групи записуємо в масиви та розраховуємо результуючі конверсії та розміри цих вибірок:

```
control_group = []
```

```
variant_group = []

control_conversion = np.sum(control_group)
control_size = len(control_group)
control_conversion_rate = control_conversion / control_size

variant_conversion = np.sum(variant_group)
variant_size = len(variant_group)
variant_conversion_rate = variant_conversion / variant_size
```

Та аналізуємо результат тесту, записуючи його в змінну verdict:

```
alpha = 0.05

z_score = stats.norm.ppf(1 - alpha / 2)
pooled_conversion_rate = (control_conversion + variant_conversion) /
(control_size + variant_size)
standard_error = np.sqrt(pooled_conversion_rate * (1 -
pooled_conversion_rate) * (1 / control_size + 1 / variant_size))
test_statistic = (variant_conversion_rate - control_conversion_rate) /
standard_error

p_value = 2 * (1 - stats.norm.cdf(np.abs(test_statistic)))

N_real = None

if N_real > control_size:
    if p_value < alpha:
        if control_conversion_rate > 0:
            verdict = "B"
        else:
            verdict = "A"
    else:
        verdict = "no difference"
else:
    verdict = "None"
```

Таким чином, ми отримуємо автоматичний розрахунок та аналіз тесту проведеного методом fixed-based horizon.

2.1.2. Автоматизація frequentist interference

Для початку, опишемо псевдокодом загальний алгоритм проведення тесту:

```
if Pvalue < a then
  if power > 1 - B then
    if Lift > 0 then
      verdict = «B»
    else
      verdict = «A»
    end if
  else
    verdict = «None»
  end if
else
  if power > 1 - B then
    verdict = «no difference»
  end if
end if
```

Ми починаємо з зберігання даних контрольної та тестової груп:

```
control_group =[]
variant_group =[]
```

Далі ми розраховуємо попередні показники для обох груп, такі як загальна кількість конверсій, розмір вибірки та коефіцієнт конверсії:

```
control_conversion = np.sum(control_group)
control_size = len(control_group)
control_conversion_rate = control_conversion / control_size


variant_conversion = np.sum(variant_group)
variant_size = len(variant_group)
variant_conversion_rate = variant_conversion / variant_size
```

Проводимо статистичну перевірку за допомогою двопропорційного z-тесту. Задаємо рівень значущості (альфа) та обчислюємо z-критерій:

```
alpha = 0.05
power = 0.8
minimum_detectable_effect = 0.05
beta = 0.2


z_score = stats.norm.ppf(1 - alpha / 2)
p_value = stats.norm.cdf(z_score)
```

Та аналізуємо результати:

```
if p_value < alpha:
```

```
    if power > 1 - beta:
        if minimum_detectable_effect > 0:
            verdict = "B"
        else:
            verdict = "A"
    else:
        verdict = "None"
else:
    if power > 1 - beta:
        verdict = "no difference"
```

Таким чином, ми отримуємо автоматичний розрахунок та аналіз тесту проведеного методом frequentist interference.

2.1.3. Автоматизація sequential sampling

Для початку, опишемо псевдокодом загальний алгоритм проведення тесту:

```
if св - са > 2*sqrt(N) then
  verict = «B»
end if
if св + са > N then
  verdict = «no difference»
else
  verdict = «None»
end if,
```

де св, са – кількість конверсій в кожній з груп відповідно, N – необхідний розмір вибірки.

Ми починаємо з зберігання даних контрольної та тестової груп:

```
control_group =[]
variant_group =[]
```

Далі ми розраховуємо попередні показники для обох груп, такі як необхідна кількість конверсій для кожної з груп:

```
alpha = 0.05
power = 0.8
baseline_conversion = control_conversion_rate
minimum_detectable_effect = 0.05
```

```
   z_score = stats.norm.ppf(1 - alpha / 2)
   p_pool = (control_conversion + variant_conversion) / (control_size +
variant_size)
   sample_size = np.ceil((2 * (z_score + stats.norm.ppf(power))**2) * (p_pool *
(1 - p_pool)) / minimum_detectable_effect**2)
```

Та аналізуємо результати:

```
   N = sample_size

   if variant_conversion - control_conversion > 2 * np.sqrt(N):
       verdict = "B"
   elif variant_conversion + control_conversion > N:
       verdict = "no difference"
   else:
       verdict = "None"
```

Таким чином, ми отримуємо автоматичний розрахунок та аналіз тесту проведеного методом sequential testing.

2.1.4. Автоматизація Bayesian testing

Загальний алгоритм проведення Байєсівського тесту:

1. Розрахувати апріорний розподіл
2. Почати експеримент
3. Періодично розраховувати статистики $c_A\ n_A\ c_B\ n_B$,

де c_A, c_B – кількість користувачів, які виконали цільову дію;

n_A, n_B – кількість користувачів у A чи B групі відповідно.

4. порахувати $E[\mathscr{L}(A)]$ та $E[\mathscr{L}(B)]$; визначити множину Q = $\{x : E[\mathscr{L}x] < \varepsilon\}$
5. Якщо |Q| = 0, продовжувати експеримент. Інакше, обрати елемент з Q, який буде переможцем.

Ми будемо використовувати бета-розподіл як спряжений апріорний розподіл. Для вибору апріорного розподілу буде використано наступну формулу [4]:

$prior = B(\alpha, \beta)$,

де n - розмір вибірки,

c - кількість конверсій,

$\alpha = c + 1$,

$\beta = n - c + 1$

За допомогою динамічного підбору n можна підібрати оптимальне α та β.

Ми починаємо з зберігання даних контрольної та тестової груп:

```
control_group =[]
variant_group =[]
```

Далі ми розраховуємо попередні показники для обох груп, такі як необхідна кількість конверсій для кожної з груп:

```
control_conversion = np.sum(control_group)
control_size = len(control_group)

variant_conversion = np.sum(variant_group)
variant_size = len(variant_group)
```

Проводимо тест, збираючи результати з заданою періодичністю (наприклад, кожні 100 користувачів у вибірці):

```
epsilon = 0.01

def calculate_expected_loss(c, n):
    alpha = c + 1
    beta = n - c + 1
    expected_loss = 1 - stats.beta.mean(alpha, beta)
    return expected_loss

def choose_winner():
    loss_control = calculate_expected_loss(control_conversion, control_size)
    loss_variant = calculate_expected_loss(variant_conversion, variant_size)
    Q = set()

    if loss_control < epsilon:
        Q.add('A')
    if loss_variant < epsilon:
        Q.add('B')
```

```
        if len(Q) == 0:
            return None
        else:
            return Q.pop()

    for i in range(0, len(control_group), 100):
        control_conversion = np.sum(control_group[:i+100])
        control_size = i + 100

        variant_conversion = np.sum(variant_group[:i+100])
        variant_size = i + 100

        winner = choose_winner()
        if winner is not None:
            break
```

Таким чином, ми отримуємо автоматичний розрахунок та аналіз тесту проведеного методом Bayesian testing.

2.2. Порівняльний аналіз

Для порівняння методів будуть розглянуті такі фактори, як статистична точність, ефективність, пристосованість до різних сценаріїв і простота інтерпретації.

Спершу, порівняємо методи fixed-based horizon та frequentist interference. Оскільки з розділу 1 розуміємо, що frequentist interference є більш динамічним підходом і в найгіршому випадку буде тривати настільки ж довго, як fixed-based horizon, робимо висновок, що майже у всіх випадках краще користуватись frequentist interference. Проте, якщо бізнесу потрібно чітко наперед розуміти тривалість та вартість тесту – тоді обираємо fixed-based horizon.

Наступним кроком порівняємо fixed-based horizon з sequential sampling. Розглянемо розраховані N та $d*$ для заданих рівнів значущості, потужностей та підйомів. За базову конверсію взято 1%. Стовпчик $E[N|HA]$ - це очікувана

кількість перетворень, необхідних для завершення тесту за альтернативної гіпотези. Стовпчик $NZ/100$ є порівнянною кількістю загальних конверсій, очікуваних за fixed-based horizon. Стовпчик Savings математично = 1-$E[N|HA]/(NZ/100)$, тобто відсоток зменшення розміру вибірки при порівнянні sequential sampling тесту з тестом з фіксованою вибіркою.

α	$1-\beta$	Lift	N	$d*$	$E[N\|HA]$	$NZ/100$	Savings
0.05	0.8	50%	170	26	120	106	−13.9%
		10%	2,922	106	2,055	2,489	17.40%
0.01	0.8	50%	262	42	198	169	−17.4%
		10%	4,644	176	3,491	4,026	13.30%

Таблиця 1 - Порівняння fixed-based horizon та sequential sampling

Бачимо, що sequential testing працює набагато краще, коли ефект присутній. Він демонструє найкращі результати при виявленні невеликих підйомів. Наприклад, якщо розмір вибірки було обрано для виявлення 10% підйому, очікувана економія за альтернативною гіпотезою сягає 17,4%. Але при виявленні дуже високого (50%) підйому будь-яка економія по суті зводиться нанівець, і в кінцевому підсумку це вимагає більшої кількості спостережень, ніж тест з фіксованою вибіркою.

Також порівняємо sequential testing та Bayesian testing підходи.

test_id	c_conv	t_conv	p	power	lift	P(A>B)	P(B>A)	L(A)	L(B)	z_verdict	b_verdict
test1	0.04	0.04	0.112	0.349	18.04	0.06	0.955	0.01	0	None	B
test2	0.68	0.67	0.102	0.347	-1.03	0.95	0.05	0	0	None	A
test3	0.76	0.67	0	1	-11.29	1	0	0	0.09	A	A

Таблиця 2 - Порівняння sequential testing та Bayesian testing

Бачимо, що майже у всіх випадках метод Bayesian testing дозволив раніше відповісти на питання яка група є більш ефективною.

Отже, враховуючи порівняння методів, було виведено оптимальне рішення який метод використовувати при тому чи іншому випадку у бізнесі:

Fixed-based horizon:

- якщо необхідно наперед чітко знати тривалість та вартість тесту

Frequentist interference:

- Continuous метрики
- Тести на високу конверсію

Sequential sampling:

- Тест на низьку конверсію
- Швидке виявлення аномалій

Bayesian testing:

- Тест на низьку конверсію
- Пропорційні тести
- Негативні тести (тести, що очікувано мають погіршити цільову метрику)

Висновки до розділу 2

В даному розділі були розроблені алгоритми для проведення A/B тестування з використанням різних методів, таких як Fixed-based horizon, Frequentist interference, Sequential sampling і Bayesian testing. Кожен з цих методів має свої особливості і використовується для вирішення певних завдань у проведенні A/B тестів. Відповідно до порівняльного аналізу цих методів було описано випадки у бізнесі та у яких з них рекомендується використовувати той чи інший метод.

3 РОЗРОБКА ДОДАТКА

3.1. Огляд існуючих рішень

В ході дослідження предметної області було проаналізовано існуючі аналоги для автоматизації розрахунку та аналізу перевірки статистичних гіпотез для A/B тестування. Всі існуючі застосунки у відкритому доступі вирішують лише проблему розрахунку попередніх метрик (необхідний розмір вибірки, кількість конверсій, тощо). Поспілкувавшись з компаніями, які використовують A/B тестування для прийняття рішень було виявлено, що задля вирішення задачі автоматичного аналізу вони використовують самостійно розроблені системи. Пояснюється це тим, що кожен бізнес використовує свій спосіб та інструменти для збереження даних дій користувачів, відповідно, це забезпечує відсутність стандартизованого вигляду даних на ринку та складність створення централізованої системи автоматичного аналізу тестів для різноманітних бізнесів.

Найбільш популярним додатком для розрахунку попередніх метрик тестів є веб-застосунок Evan Miller [5]. У ньому можливо порахувати необхідний розмір вибірки для Fixed-based horizon тестів та потрібну кількість конверсій для Sequential sampling.

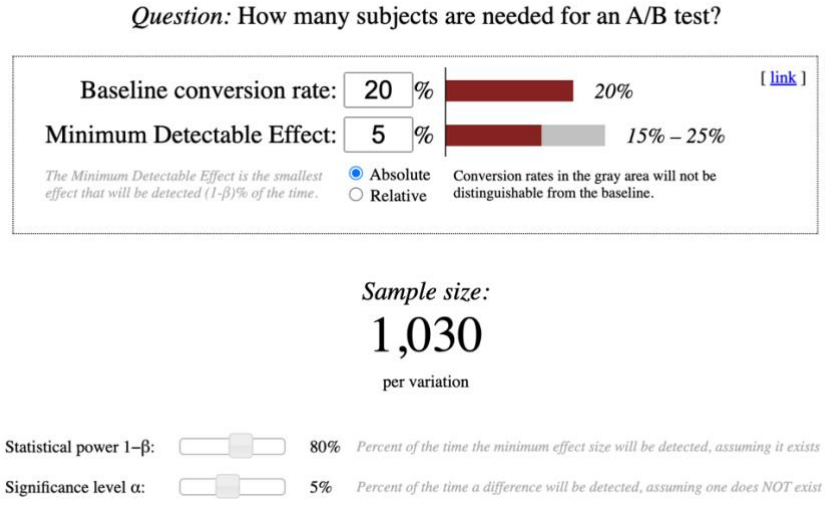
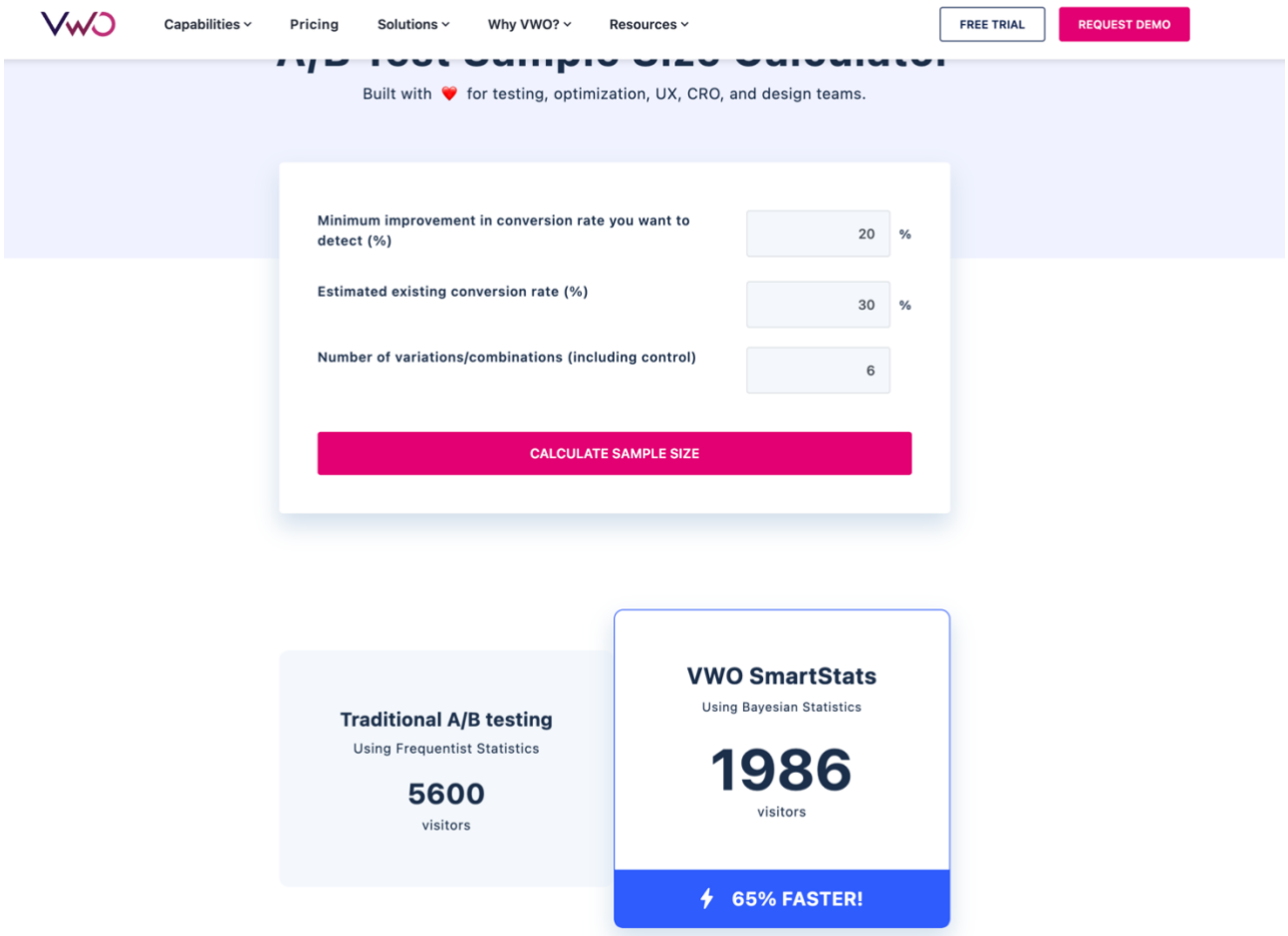


Рисунок 2 - Калькулятор Evan Miller

На веб-застосунку VMO також розраховують приблизну кількість користувачів, яка буде необхідна для проведення тесту за допомогою Bayesian методу [9].



Інші аналоги на ринку, а зокрема ABTasty [6], CXL [7], Optimizely [8], пропонують лише розрахунок для fixed-based horizon методу.

Проаналізувавши існуючі рішення стає цілком очевидним, що автоматизація аналізу результатів різноманітних методів A/B тестування залишається невирішеною задачею, що в черговий раз підтверджує актуальність розробки такої системи.

3.2. Вибір технологій та інструментів розробки

Важливо було правильно підібрати інструменти та технології для розробки додатку, щоб успішно реалізувати весь функціонал, який був передбачений для нього. Були обрані наступні технології та інструменти:

- Інтегроване середовище розробки (IDE) JetBrains, WebStorm, було обрано в якості основного середовища для розробки. Завдяки повній підтримці JavaScript, TypeScript, Python, HTML та CSS, WebStorm пропонує стабільне та багатофункціональне рішення. Його редактор коду та безперешкодна інтеграція з системою контролю версій покращують процес розробки, дозволяючи ефективно працювати та досягати швидшого результату.
- React: Для проектування користувацьких інтерфейсів було обирано добре відому JavaScript-бібліотеку React для фронтенд-розробки. React є одним з найкращих пакетів для створення динамічних та інтерактивних онлайн-додатків завдяки своїй компонентній архітектурі, ефективному рендерингу віртуального DOM та надійній екосистемі. React дозволяє створювати багаторазові компоненти інтерфейсу користувача, керувати управлінням станами та гарантувати оптимальну продуктивність, швидко оновлюючи лише необхідні елементи інтерфейсуА [10].
- Для управління HTTP-запитами та API-з'єднаннями було обрано простий у використанні JavaScript-пакет Axios. Axios надає простий у використанні функціонал для передачі та отримання даних, що спрощує процес виконання асинхронних HTTP-з'єднань. Це надійний варіант для управління клієнт-серверним з'єднанням завдяки своїм характеристикам, включаючи автоматичне скасування запитів і управління помилками [11].
- HTML і CSS - це ключові технології організації та стилізації веб-контенту. Вони були обрані як доповнення до React-у для побудови графічного інтерфейсу нашого додатку. У той час як CSS дає нам можливість змінювати зовнішній вигляд і стиль веб-сторінок, HTML диктує їх структуру і зміст. Разом вони пропонують фундаментальні компоненти для естетично привабливого та адаптивного користувацького інтерфейсу.
- Flask: Для розробки бекенду було обрано Flask, компактний та адаптивний веб-фреймворк Python. Простота та розширюваність Flask дозволяє створювати RESTful API та ефективно керувати функціональністю на

стороні сервера. Його модульна структура та ретельна документація полегшують розробку, а взаємодія з відомими бібліотеками та інструментами спрощує реалізацію різноманітного функціоналу [12].

- MongoDB: MongoDB було обрано як NoSQL базу даних через її масштабованість, адаптивність та простоту взаємодії з Flask. Структуровані дані можна ефективно зберігати та вилучати за допомогою формату даних на основі документів MongoDB, який також підтримує горизонтальне масштабування для задоволення зростаючих вимог програми. Це хороший варіант для зберігання та обробки даних завдяки сумісності з документами, що нагадують JSON, та чудовим можливостям запитів [13].
- SendGrid: Додаток був інтегрований з SendGrid, провайдером доставки електронної пошти, для надсилання листів про завершення A/B тесту. Завдяки надійній інфраструктурі та зручному API SendGrid можна легко надсилати користувачам кастомізовані електронні листи [14].
- APScheduler: Було підключено APScheduler, фреймворк для планування фонових задач на Python, для автоматизації повторюваних завдань. Широкі можливості планування, які надає APScheduler, такі як тригери на основі інтервалів або cron, дозволяють нам створювати фонові операції, такі як погодинний аналіз тестів [15].

Було створено потужне та ефективне середовище розробки, ретельно обираючи дані технології та інструменти. React, Axios, HTML та CSS стали основою для створення сучасного та інтерактивного користувацького інтерфейсу, а WebStorm - інтуїтивно зрозумілим IDE для безперешкодного програмування. Завдяки Flask, MongoDB, SendGrid та APScheduler було можливо створити необхідні бекенд-операції, зберігати дані, надсилати електронну пошту та виконувати фонові задачі. Разом ці технології та інструменти дали можливість створити додаток, який є надійним та масштабованим.

3.3. Розробка архітектури та інтерфейсу програмного додатка

Архітектура додатку відповідає клієнт-серверній моделі, де фронтенд реалізований за допомогою React.js, а бекенд - за допомогою Flask, веб-фреймворку на Python.

Загальна логіка застосунку наступна:

- Користувач за допомогою користувацького інтерфейсу вибирає цікавий йому метод тестування та вхідні дані для розрахунку цього тесту.
- Користувач має можливість запустити тест – в цей момент в базі даних створюється новий документ з інформацією про цей тест.
- Кожної години фонова задача перевіряє всі відкриті тести та рахує для кожного тригери зупинки.
- Якщо тригер спрацьовує – ми закриваємо тест та надсилаємо користувачу електронну пошту з інформацією про закриття тесту.

Розглянемо логіку створення користувацького інтерфейсу. Основними функціональними вимогами було:

- Простота за зручність користування
- Можливість користувача вибрати необхідний йому метод тестування
- Можливість вказати попередні метрики, необхідні для тесту. Наприклад, статистичну потужність, базову конверсію та мінімальний бажаний ефект для розрахунку необхідної кількості користувачів для Fixed-based horizon методу тестування.
- Можливість запустити тест за умови, якщо всі потрібні тесту поля заповнені.

Після того, як користувач потрапляє на платформу, він бачить перед собою основну форму розрахунку для тестів:

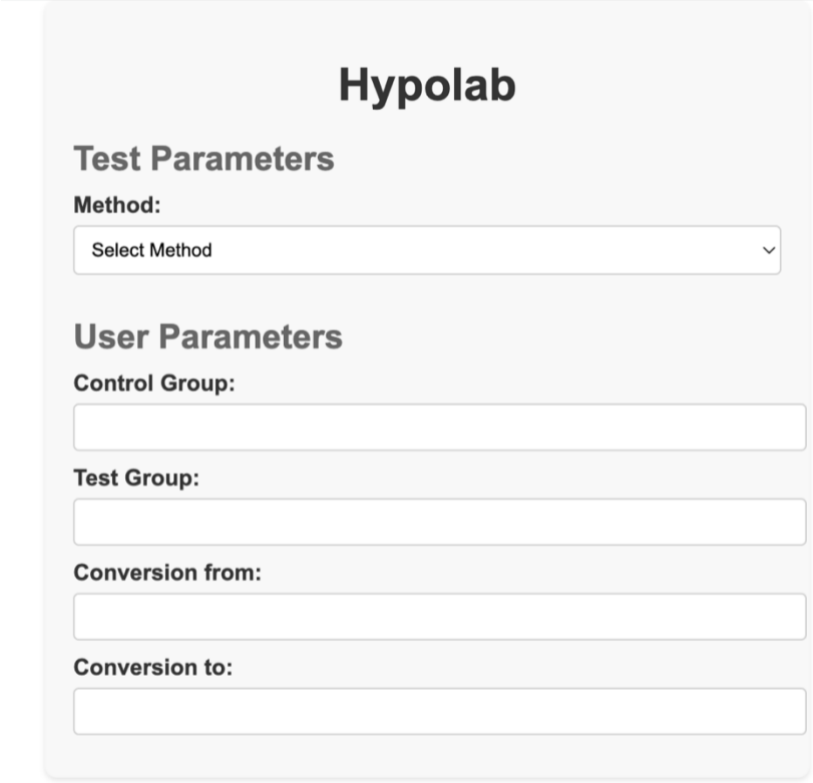


Рисунок 3 - Початкова форму після потрапляння на платформу

Оскільки користувач ще не обрав ніякого методу тестування, йому недоступні кнопки розрахунку та запуску. Форма розрахунку реалізована за допомогою функціонального React-компонента з назвою App, який слугує основним компонентом додатку. Він включає декілька змінних стану для управління користувацьким вводом та різні функції для обробки обчислень і взаємодії з сервером.

Змінні стану: Хук useState використовується для оголошення та ініціалізації декількох змінних стану, таких як method, baselineConversion, minimumDetectableEffect, statisticalPower, significanceLevel, epsilon, sampleSize, testStarted, controlGroup, testGroup, conversionFrom і conversionTo. Ці змінні зберігають введені користувачем дані, розрахований розмір вибірки та статус тесту.

```
const [method, setMethod] = useState('');
const [baselineConversion, setBaselineConversion] = useState('');
const [minimumDetectableEffect, setMinimumDetectableEffect] = useState('');
const [statisticalPower, setStatisticalPower] = useState(0.8);
const [significanceLevel, setSignificanceLevel] = useState(0.05);
```

```
const [epsilon, setEpsilon] = useState(0.01);
const [sampleSize, setSampleSize] = useState('');
const [testStarted, setTestStarted] = useState(false);
const [controlGroup, setControlGroup] = useState('');
const [testGroup, setTestGroup] = useState('');
const [conversionFrom, setConversionFrom] = useState("");
const [conversionTo, setConversionTo] = useState("");
```

Відрендерений JSX: Метод рендеру компонента повертає JSX, що представляє користувацький інтерфейс програми. Він включає різні елементи форми, такі як мітки, поля введення, кнопки, а також умовний рендеринг на основі обраного методу та інших умов. Введені користувачем дані прив'язуються до відповідних змінних стану, і визначені обробники подій, які оновлюють змінні стану після взаємодії з користувачем.

Створення форми для вибору методу тестування:

```
<h2>Test Parameters</h2>
            <label>Method:</label>
            <select value={method} onChange={handleMethodChange}>
                <option value="">Select Method</option>
                <option value="fixed">Fixed-based Horizon</option>
                <option value="sequential">Sequential Sampling</option>
                <option value="frequentist">Frequentist Inference</option>
                <option value="bayesian">Bayesian Testing</option>
            </select>
```

Наступним кроком ми створюємо поля для вводу користувача за умови, що він вибрав якийсь конкретний метод тестування:

```
{method && (
                <>
                    <label>Baseline Conversion:</label>
                    <input
                        type="number"
                        step="0.01"
                        value={baselineConversion}
                        onChange={(e) => setBaselineConversion(e.target.value)}
                    />
```

```
                    <label>Minimum Detectable Effect:</label>
                    <input
                        type="number"
                        step="0.01"
                        value={minimumDetectableEffect}
                        onChange={(e) =>
setMinimumDetectableEffect(e.target.value)}
                    />
                </>
            )}
```

Baseline Conversion та Minimum Detectable Effect – це параметри, які будуть використуватися у всіх передбачених методах тестування. Проте, параметри статистичної потужності та рівня значущості будемо використовувати лише у Fixed-based horizon та Frequentist interference. Саме тому створюємо додаткову перевірку та поля вводу для користувача:

```
{method === "fixed" || method === "frequentist" ? (
                <>
                    <div className="parameter-container">
                        <div className="parameter">
                            <label>Statistical Power (1-β):</label>
                            <input
                                type="range"
                                value={statisticalPower}
                                min="0"
                                max="1"
                                step="0.01"
                                onChange={(e) =>
setStatisticalPower(parseFloat(e.target.value))}
                            />
                            <span>{statisticalPower}</span>
                        </div>
                        <div className="parameter">
                            <label>Significance Level (α):</label>
                            <input
                                type="range"
                                value={significanceLevel}
```

```
                                min="0"
                                max="1"
                                step="0.01"
                                onChange={(e) =>
setSignificanceLevel(parseFloat(e.target.value))}
                            />
                            <span>{significanceLevel}</span>
                        </div>
                    </div>
                    {method === "fixed" && baselineConversion &&
minimumDetectableEffect && statisticalPower && significanceLevel && (
                        <button onClick={calculateSampleSize}>Calculate Sample
Size</button>
                    )}
                </>
```

При виборі Байєсівського тестування додаємо можливість ввести параметр epsilon:

```
: method === "bayesian" ? (
                <>
                    <label>Epsilon:</label>
                    <input
                        type="number"
                        step="0.01"
                        value={epsilon}
                        onChange={(e) =>
setEpsilon(parseFloat(e.target.value))}
                    />
                </>
            ) : (
                <></>
            )}
```

Також створюємо поля для вводу унікальних ідентифікаторів тесту, щоб при зборі даних було зрозуміло, на що орієнтуватись:

```
<h2>User Parameters</h2>
            <label>Control Group:</label>
```

```
            <input
                type="text"
                value={controlGroup}
                onChange={(e) => setControlGroup(e.target.value)}
            />

            <label>Test Group:</label>
            <input
                type="text"
                value={testGroup}
                onChange={(e) => setTestGroup(e.target.value)}
            />

            <label>Conversion from:</label>
            <input
                type="text"
                value={conversionFrom}
                onChange={(e) => setConversionFrom(e.target.value)}
            />

            <label>Conversion to:</label>
            <input
                type="text"
                value={conversionTo}
                onChange={(e) => setConversionTo(e.target.value)}
            />

            {(method === "fixed" || method === "frequentist") && sampleSize &&
controlGroup && testGroup && conversionFrom && conversionTo && (
                <button onClick={startTest}>Start Test</button>
            )}

            {(method === "sequential" || method === "bayesian") &&
baselineConversion && minimumDetectableEffect && controlGroup && testGroup &&
conversionFrom && conversionTo && (
                <button onClick={startTest}>Start Test</button>
            )}
            {testStarted && (
```

```
            <label>Test successfully started</label>
        )}
```

Функція calculateSampleSize викликається, коли користувач натискає кнопку "Обчислити розмір вибірки". Вона надсилає HTTP POST-запит на сервер за допомогою Axios. Запит містить параметри тесту (наприклад, метод, мінімальний бажаний ефект, статистичну потужність, рівень значущості) в тілі запиту. Отримавши відповідь, функція витягує розрахований розмір вибірки і відповідно оновлює змінну стану sampleSize.

```
function calculateSampleSize() {
        axios({
            method: "POST",
            url:"/sample_size_calculation",
            data: {
                method,
                baselineConversion,
                minimumDetectableEffect,
                statisticalPower,
                significanceLevel
            }
        })
            .then((response) => {
                const sampleSize = response.data.sampleSize;
                setSampleSize(sampleSize)
            }).catch((error) => {
            if (error.response) {
                console.log(error.response)
                console.log(error.response.status)
                console.log(error.response.headers)
            }
        })}
```

Функція startTest запускається, коли користувач натискає кнопку "Почати тест". Вона також надсилає HTTP POST-запит на сервер за допомогою Axios,

включаючи різні параметри тесту в тілі запиту. Після отримання відповіді функція оновлює змінну стану testStarted на основі даних відповіді.

```
function startTest() {
        axios({
            method: "POST",
            url:"/start_test",
            data: {
                method,
                baselineConversion,
                minimumDetectableEffect,
                statisticalPower,
                significanceLevel,
                sampleSize,
                controlGroup,
                testGroup,
                conversionFrom,
                conversionTo,
                epsilon
            }
        })
            .then((response) => {
                const testStarted = response.data.started;
                setTestStarted(testStarted)
            }).catch((error) => {
            if (error.response) {
                console.log(error.response)
                console.log(error.response.status)
                console.log(error.response.headers)
            }
        })}
```

Функція handleMethodChange викликається, коли користувач вибирає метод зі спадного меню. Вона оновлює змінну стану методу та скидає інші відповідні змінні стану до початкових значень.

```
function handleMethodChange(e) {
        const selectedMethod = e.target.value;
```

```
    setMethod(selectedMethod);
    setBaselineConversion('');
    setMinimumDetectableEffect('');
    setStatisticalPower(0.8);
    setSignificanceLevel(0.05);
    setSampleSize('');
    setTestStarted(false);
    setControlGroup('');
    setTestGroup('');
    setConversionFrom('');
    setConversionTo('');
    setEpsilon(0.01);
}
```

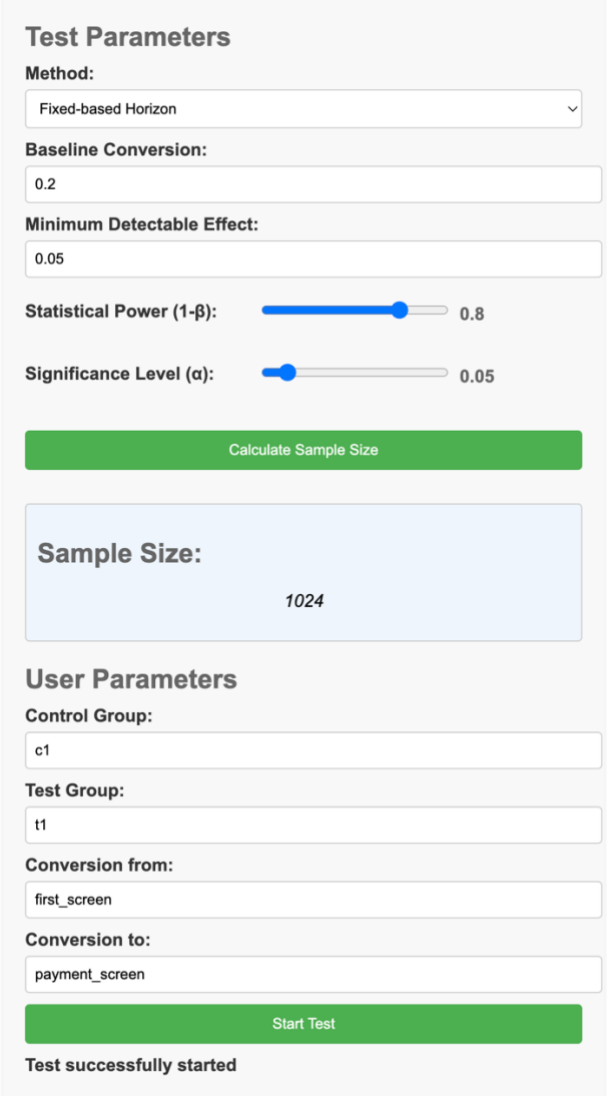


Рисунок 4 - Форма вводу при виборі Fixed-based horizon

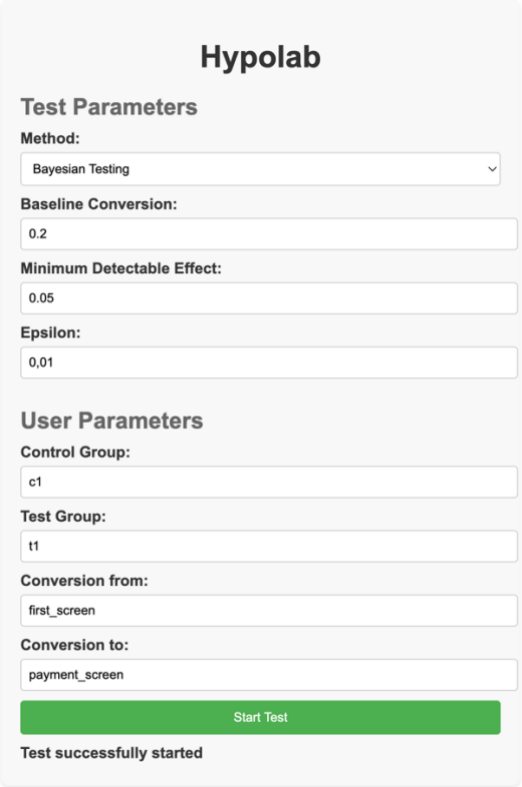


Рисунок 5 - Форма вводу при виборі Bayesian testing

Розглядаючи реалізацію бекенд-частини, маємо 2 основних роути, до яких звертаються функції startTest та calculateSampleSize:

- /sample_size_calculation: Цей роут очікує POST-запит з параметрами, пов'язаними з методом тестування (фіксованим або іншим). Він обчислює розмір вибірки на основі параметрів і повертає його у відповідь.

```
@app.route('/sample_size_calculation', methods=['POST'])
def sample_size_calculation():
    method = str(request.json['method'])
    baseline_conversion = float(request.json['baselineConversion'])
    minimum_detectable_effect = float(request.json['minimumDetectableEffect'])
    statistical_power = float(request.json['statisticalPower'])
    significance_level = float(request.json['significanceLevel'])

    if method == 'fixed':
        sample_size =
calculate_sample_size_for_fixed_based_horizon(baseline_conversion,
minimum_detectable_effect, significance_level, statistical_power)
    else:
        sample_size = -1
```

```
    response = {'sampleSize': sample_size}
    return response
```

Основна логіка розрахунку знаходиться у функції calculate_sample_size_for_fixed_based_horizon:

```
def calculate_sample_size_for_fixed_based_horizon(baseline_conversion,
minimum_detectable_effect, significance_level, statistical_power):
    p1 = baseline_conversion
    p2 = baseline_conversion * (1 + minimum_detectable_effect)
    pool_sd = np.sqrt((p1 * (1 - p1) + p2 * (1 - p2)) / 2)
    z_score = stats.norm.ppf(1 - significance_level / 2)
    sample_size = np.ceil(
        (2 * (z_score + stats.norm.ppf(statistical_power)) ** 2) /
minimum_detectable_effect ** 2 * pool_sd ** 2)
    return sample_size
```

- /start_test: Цей маршрут очікує POST-запит з параметрами для запуску нового тесту. Він вставляє дані тесту в базу даних MongoDB і повертає відповідь із зазначенням того, чи був тест успішно запущений чи ні.

```
@app.route('/start_test', methods=['POST'])
def start_test():
    try:
        method = str(request.json['method'])
        baseline_conversion = float(request.json['baselineConversion'])
        minimum_detectable_effect =
float(request.json['minimumDetectableEffect'])
        control_group_marker = str(request.json['controlGroup'])
        test_group_marker = str(request.json['testGroup'])
        conversion_from = str(request.json['conversionFrom'])
        conversion_to = str(request.json['conversionTo'])

        date_start = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

        item_id =
f"{date_start}_{method}_{control_group_marker}_{test_group_marker}_{conversion_
from}_{conversion_to}"
```

```
        item = {
                '_id': item_id,
                'started': date_start,
                'method': method,
                'baseline_conversion': baseline_conversion,
                'minimum_detectable_effect': minimum_detectable_effect,
                'control_group_marker': control_group_marker,
                'test_group_marker': test_group_marker,
                'conversion_from': conversion_from,
                'conversion_to': conversion_to,
                'control_data': [],
                'test_data': [],
                'finished': False
            }

        if method =='fixed' or method == 'frequentist':
            item['statistical_power'] = float(request.json['statisticalPower'])
            item['significance_level'] =
float(request.json['significanceLevel'])
            if method == 'fixed':
                item['sample_size'] = float(request.json['sampleSize'])
        elif method == 'bayesian':
            item['epsilon'] = float(request.json['epsilon'])

        db.tests_data.insert_one(item)
        return {'started': True}
    except Exception as e:
        return {'started': False}
```

Для роботи з даними було вибрано NoSQL базу даних MongoDB. У базі зберігаються сутності кожного тесту з полями, як дата, коли тест було запущено, тип тесту, базова конверсія, тощо.

Рисунок 6 - Приклад документу в MongoDB

APScheduler використовується для планування завдання, яке запускається щогодини (функція hourly_check) для перевірки незавершених тестів у базі даних. Усередині функції hourly_check він отримує незавершені тести з бази даних і виконує обчислення на основі методу тестування (фіксований, частотний, послідовний або байєсівський). Для кожного тесту функція обчислює коефіцієнти конверсії, виконує статистичні розрахунки і перевіряє результат тесту.

```
@scheduler.task('interval', id='hourly_check', hours=1, misfire_grace_time=900)
def hourly_check():
    tests = db.tests_data.find({'finished': False})

    for test in tests:
        method = test['method']
        control_data = test['control_data']
        test_data = test['test_data']
        minimum_detectable_effect = test['minimum_detectable_effect']
        control_group_marker = test['control_group_marker']
        test_group_marker = test['test_group_marker']
        conversion_from = test['conversion_from']
        conversion_to = test['conversion_to']

        control_conversion = np.sum(control_data)
        control_size = len(control_data)
        control_conversion_rate = control_conversion / control_size
        test_conversion = np.sum(test_data)
        test_size = len(test_data)
        test_conversion_rate = test_conversion / test_size

        test_started = test['date_start']
```

```
        item_id =
f"{test_started}_{method}_{control_group_marker}_{test_group_marker}_{conversio
n_from}_{conversion_to}"

        subject = f"Test finished: {item_id}"
        message = f"Test method: {method}<br>" \
                  f"Test started: {test_started}<br><br>" \
                  f"Minimum Detectable Effect: {minimum_detectable_effect}<br>"
\
                  f"Control group marker: {control_group_marker}<br>" \
                  f"Test group marker: {test_group_marker}<br>" \
                  f"Conversion from: {conversion_from}<br>" \
                  f"Conversion to: {conversion_to}<br><br>" \
                  f"Control size: {control_size}<br>" \
                  f"Control conversion rate: {control_conversion_rate}<br>" \
                  f"Test size: {test_size}<br>" \
                  f"Test conversion rate: {test_conversion_rate}<br><br>"

        alpha = test['significance_level']
        power = test['statistical_power']
        beta = 1 - power

        if method == 'fixed':
            verdict = check_verdict_for_fixed_base_horizon(test['sample_size'],
control_size, test_size, control_conversion, test_conversion,
control_conversion_rate, test_conversion_rate, alpha)

            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'frequentist':
            verdict = check_verdict_for_frequentist_interference(alpha, beta,
power, minimum_detectable_effect)
```

```
            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'sequential':
            verdict = check_verdict_for_sequential_sampling(alpha, power,
minimum_detectable_effect, control_conversion, test_conversion, control_size,
test_size)

            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'bayesian':
            verdict = check_verdict_for_bayesian_testing(test['epsilon'],
control_conversion, control_size, test_conversion, test_size)

            if verdict is not None:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)
```

Якщо тест має групу-переможця, він оновлює статус тесту в базі даних, створює повідомлення електронної пошти і надсилає його за допомогою функції send_email.

```
    def send_email(mailApiKey, mail_from, mail_to, subject, message_text):
        mess = Mail(
            from_email=mail_from,
            to_emails=mail_to,
```

```
            subject=subject,
            html_content=message_text
        )
        sg = SendGridAPIClient(mailApiKey)
        response = sg.send(mess)
```

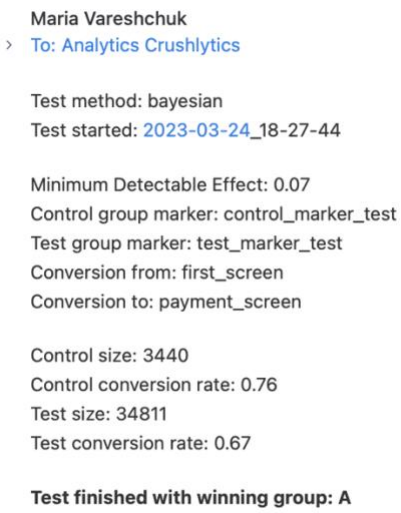


Рисунок 7 - Приклад електронного листа про завершення тесту

Файл calculations.py містить різні функції, які використовуються для різних типів розрахунків, пов'язаних з A/B тестуванням. До них відносяться розрахунок розміру вибірки для фіксованого горизонту, перевірка тестового вердикту для різних методів тестування (фіксованого, частотного, послідовного, байєсівського), а також розрахунок очікуваних втрат і вибір переможця для байєсівського тестування.

Функція перевірки завершення тесту за методом fixed-based horizon:

```
    def check_verdict_for_fixed_base_horizon(sample_size, control_size,
test_size, control_conversion, test_conversion, control_conversion_rate,
test_conversion_rate, alpha):
        pooled_conversion_rate = (control_conversion + test_conversion) /
(control_size + test_size)
        standard_error = np.sqrt(pooled_conversion_rate * (1 -
pooled_conversion_rate) * (1 / control_size + 1 / test_size))
        test_statistic = (test_conversion_rate - control_conversion_rate) /
standard_error

        p_value = 2 * (1 - stats.norm.cdf(np.abs(test_statistic)))
```

```
        if control_size >= sample_size and test_size >= sample_size:
            if p_value < alpha:
                if control_conversion_rate > 0:
                    verdict = "B"
                else:
                    verdict = "A"
            else:
                verdict = "no difference"
        else:
            verdict = None
        return verdict
```

Функція перевірки завершення тесту за методом frequentist interference:

```
def check_verdict_for_frequentist_interference(alpha, beta, power,
minimum_detectable_effect):
    z_score = stats.norm.ppf(1 - alpha / 2)
    p_value = stats.norm.cdf(z_score)

    if p_value < alpha:
        if power > 1 - beta:
            if minimum_detectable_effect > 0:
                verdict = "B"
            else:
                verdict = "A"
        else:
            verdict = None
    else:
        if power > 1 - beta:
            verdict = "no difference"
        else:
            verdict = None
    return verdict
```

Функція перевірки завершення тесту за методом Sequential sampling:

```
def check_verdict_for_sequential_sampling(alpha, power,
minimum_detectable_effect, control_conversion, test_conversion, control_size,
test_size):
```

```
    pooled_conversion_rate = (control_conversion + test_conversion) /
(control_size + test_size)
    z_score = stats.norm.ppf(1 - alpha / 2)
    sample_size = np.ceil(
        (2 * (z_score + stats.norm.ppf(power)) ** 2) * (
                    pooled_conversion_rate * (1 - pooled_conversion_rate)) /
minimum_detectable_effect ** 2)

    if test_conversion - control_conversion > 2 * np.sqrt(sample_size):
        verdict = "B"
    elif test_conversion + control_conversion > sample_size:
        verdict = "no difference"
    else:
        verdict = None
    return verdict
```

Функція перевірки завершення тесту за методом Bayesian testing:

```
def calculate_expected_loss(c, n):
    alpha = c + 1
    beta = n - c + 1
    expected_loss = 1 - stats.beta.mean(alpha, beta)
    return expected_loss


def choose_winner(control_conversion, control_size, variant_conversion,
variant_size, epsilon):
    loss_control = calculate_expected_loss(control_conversion, control_size)
    loss_variant = calculate_expected_loss(variant_conversion, variant_size)
    Q = set()

    if loss_control < epsilon:
        Q.add('A')
    if loss_variant < epsilon:
        Q.add('B')

    if len(Q) == 0:
        return None
```

```
    else:
        return Q.pop()


def check_verdict_for_bayesian_testing(epsilon, control_conversion,
control_size, test_conversion, test_size):
    return choose_winner(control_conversion, control_size, test_conversion,
test_size, epsilon)
```

Висновки до розділу 3

В результаті розробки архітектури та інтерфейсу програмного додатку було створено надійну та зручну систему для розрахунку та автоматичного аналізу A/B тестів. Архітектура відповідає модульному підходу, з чітким розподілом обов'язків між різними компонентами. Інтерфейс програми розроблений таким чином, щоб бути інтуїтивно зрозумілим і зручним для користувача.

Загалом, архітектура та інтерфейс програмного додатку були ретельно продумані та розроблені, щоб сприяти ефективному та результативному процесу тестування, дозволяючи користувачам приймати рішення на основі даних та отримувати цінну інформацію зі своїх експериментів.

ВИСНОВКИ

У даній роботі було проведено дослідження різноманітних методів перевірки статистичних гіпотез для A/B тестування на веб-застосунках та мобільних додатках. Для цього, спершу, були описані фундаментальні концепції A/B-тестування, які передбачають порівняння двох варіацій додатку для визначення їхнього впливу на поведінку користувачів та ключові показники. До них відносяться перевірка гіпотез, довірчі інтервали, p-значення, статистична потужність і розрахунок розміру вибірки.

Було проаналізовано різні статистичні методи та методології, які зазвичай використовуються для аналізу результатів тестування. Методи, що були розглянуті: Fixed-based horizon, Frequentist interference, Sequential sampling, Bayesian testing. Було проведено порівняльний аналіз даних методів та виведено оптимальний алгоритм рішень при тому чи іншому випадку у бізнесі.

Також було розроблено веб-застосунок, який використовує дані методи A/B тестування для автоматичного проведення тестів. Створено зручний користувацький інтерфейс, що забезпечив інтуїтивно зрозумілі кроки для розрахунку розміру вибірки, запуску тесту та отримання сповіщень на електронну пошту про завершення тесту. Інтеграція статистичних розрахунків та алгоритмів дозволила автоматизувати визначення цих результатів та ідентифікацію груп-переможців.

Результатом дослідження став зручний та ефективний веб-застосунок для A/B-тестування, що спрощує процес розрахунку тесту, управління даними та сприяє ефективному та вчасному прийняттю рішень. Подальші можливості розвитку додатку включають розширення функціоналу, зокрема додавання візуалізацій та відображення історії тестів, пряма інтеграція з сховищами даних з діями користувачів. Дослідження показало, що у сучасному світі, заснованому на даних, A/B-тестування відіграє надважливу роль, дозволяючи компаніям залишатися конкурентоспроможними та приймати обґрунтовані рішення. Використовуючи принципи та практики, викладені в цій роботі, зокрема способи

автоматизації тестування, компанії можуть розкрити потенціал A/B-тестування для пришвидшення зростання, оптимізації користувацького досвіду та досягнення своїх стратегічних цілей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. William Feller, An Introduction to Probability Theory and Its Applications, Vol. 1, Wiley, 3rd edition (1968) pp. 349-351
2. William Feller, An Introduction to Probability Theory and Its Applications, Vol. 1, Wiley, 3rd edition (1968) pp. 89-91
3. C. Stucchio, Bayesian A/B Testing at VWO (2015) pp 1-10
4. Formulas for Bayesian A/B Testing [Електронний ресурс] // Evan Miller. – 2015. — Режим доступу до ресурсу: https://www.evanmiller.org/bayesian-ab-testing.html
5. Evan Miller Sample Size Calculator [Електронний ресурс] // Evan Miller. – 2015. — Режим доступу до ресурсу: https://www.evanmiller.org/ab-testing/sample-size.html
6. ABTasty Sample Size Calculator [Електронний ресурс] // ABTasty. — Режим доступу до ресурсу: https://www.abtasty.com/sample-size-calculator/
7. CXL Sample Size Calculator [Електронний ресурс] // CXL. — Режим доступу до ресурсу: https://cxl.com/ab-test-calculator/
8. Optimizely Sample Size Calculator [Електронний ресурс] // Optimizely. — Режим доступу до ресурсу: https://www.optimizely.com/sample-size-calculator/#/?conversion=3&effect=20&significance=95
9. VMO Sample Size Calculator [Електронний ресурс] // VMO. — Режим доступу до ресурсу: https://vwo.com/tools/ab-test-sample-size-calculator/
10. React Documentation [Електронний ресурс] // Meta Platforms. — Режим доступу до ресурсу: https://legacy.reactjs.org/docs/getting-started.html
11. Axios Documentation [Електронний ресурс] // Axios. — Режим доступу до ресурсу: https://axios-http.com/docs/intro
12. Flask Documentation [Електронний ресурс] // Pallets. — Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.3.x/
13. MongoDB Documentation [Електронний ресурс] // MongoDB. — Режим доступу до ресурсу: https://www.mongodb.com/docs/

14. SendGrid Documentation [Електронний ресурс] // SendGrid. — Режим доступу до ресурсу: https://docs.sendgrid.com/
15. APScheduler Documentation [Електронний ресурс] // APScheduler. — Режим доступу до ресурсу: https://apscheduler.readthedocs.io/en/3.x/

ДОДАТКИ

Додаток А. Код файлу App.js

```
import React, { useState } from 'react';
import axios from 'axios';
import './App.css';

function App() {
    const [method, setMethod] = useState('');
    const [baselineConversion, setBaselineConversion] = useState('');
    const [minimumDetectableEffect, setMinimumDetectableEffect] = useState('');
    const [statisticalPower, setStatisticalPower] = useState(0.8);
    const [significanceLevel, setSignificanceLevel] = useState(0.05);
    const [epsilon, setEpsilon] = useState(0.01);
    const [sampleSize, setSampleSize] = useState('');
    const [testStarted, setTestStarted] = useState(false);
    const [controlGroup, setControlGroup] = useState('');
    const [testGroup, setTestGroup] = useState('');
    const [conversionFrom, setConversionFrom] = useState("");
    const [conversionTo, setConversionTo] = useState("");

    function calculateSampleSize() {
        axios({
            method: "POST",
            url:"/sample_size_calculation",
            data: {
                method,
                baselineConversion,
                minimumDetectableEffect,
                statisticalPower,
                significanceLevel
            }
        })
            .then((response) => {
                const sampleSize = response.data.sampleSize;
                setSampleSize(sampleSize)
```

```
        }).catch((error) => {
        if (error.response) {
            console.log(error.response)
            console.log(error.response.status)
            console.log(error.response.headers)
        }
    })}



function startTest() {
    axios({
        method: "POST",
        url:"/start_test",
        data: {
            method,
            baselineConversion,
            minimumDetectableEffect,
            statisticalPower,
            significanceLevel,
            sampleSize,
            controlGroup,
            testGroup,
            conversionFrom,
            conversionTo,
            epsilon
        }
    })
        .then((response) => {
            const testStarted = response.data.started;
            setTestStarted(testStarted)
        }).catch((error) => {
        if (error.response) {
            console.log(error.response)
            console.log(error.response.status)
            console.log(error.response.headers)
        }
    })}
```

```
function handleMethodChange(e) {
    const selectedMethod = e.target.value;
    setMethod(selectedMethod);
    setBaselineConversion('');
    setMinimumDetectableEffect('');
    setStatisticalPower(0.8);
    setSignificanceLevel(0.05);
    setSampleSize('');
    setTestStarted(false);
    setControlGroup('');
    setTestGroup('');
    setConversionFrom('');
    setConversionTo('');
    setEpsilon(0.01);
}

return (
    <div className="App">
        <h1>Hypolab</h1>

        <h2>Test Parameters</h2>
        <label>Method:</label>
        <select value={method} onChange={handleMethodChange}>
            <option value="">Select Method</option>
            <option value="fixed">Fixed-based Horizon</option>
            <option value="sequential">Sequential Sampling</option>
            <option value="frequentist">Frequentist Inference</option>
            <option value="bayesian">Bayesian Testing</option>
        </select>

        {method && (
            <>
                <label>Baseline Conversion:</label>
                <input
                    type="number"
                    step="0.01"
```

```
                        value={baselineConversion}
                        onChange={(e) => setBaselineConversion(e.target.value)}
                    />

                    <label>Minimum Detectable Effect:</label>
                    <input
                        type="number"
                        step="0.01"
                        value={minimumDetectableEffect}
                        onChange={(e) =>
setMinimumDetectableEffect(e.target.value)}
                    />
                </>
            )}

            {method === "fixed" || method === "frequentist" ? (
                <>
                    <div className="parameter-container">
                        <div className="parameter">
                            <label>Statistical Power (1-β):</label>
                            <input
                                type="range"
                                value={statisticalPower}
                                min="0"
                                max="1"
                                step="0.01"
                                onChange={(e) =>
setStatisticalPower(parseFloat(e.target.value))}
                            />
                            <span>{statisticalPower}</span>
                        </div>
                        <div className="parameter">
                            <label>Significance Level (α):</label>
                            <input
                                type="range"
                                value={significanceLevel}
                                min="0"
                                max="1"
```

```
                                step="0.01"
                                onChange={(e) =>
setSignificanceLevel(parseFloat(e.target.value))}
                            />
                            <span>{significanceLevel}</span>
                        </div>
                    </div>
                    {method === "fixed" && baselineConversion &&
minimumDetectableEffect && statisticalPower && significanceLevel && (
                        <button onClick={calculateSampleSize}>Calculate Sample
Size</button>
                    )}
                </>
            ) : method === "bayesian" ? (
                <>
                    <label>Epsilon:</label>
                    <input
                        type="number"
                        step="0.01"
                        value={epsilon}
                        onChange={(e) =>
setEpsilon(parseFloat(e.target.value))}
                    />
                </>
            ) : (
                <></>
            )}

            {sampleSize && (
                <div className="result">
                    <h2>Sample Size:</h2>
                    <p>{sampleSize}</p>
                </div>
            )}

            <h2>User Parameters</h2>
            <label>Control Group:</label>
            <input
```

```
                type="text"
                value={controlGroup}
                onChange={(e) => setControlGroup(e.target.value)}
            />

            <label>Test Group:</label>
            <input
                type="text"
                value={testGroup}
                onChange={(e) => setTestGroup(e.target.value)}
            />

            <label>Conversion from:</label>
            <input
                type="text"
                value={conversionFrom}
                onChange={(e) => setConversionFrom(e.target.value)}
            />

            <label>Conversion to:</label>
            <input
                type="text"
                value={conversionTo}
                onChange={(e) => setConversionTo(e.target.value)}
            />

            {(method === "fixed" || method === "frequentist") && sampleSize &&
controlGroup && testGroup && conversionFrom && conversionTo && (
                <button onClick={startTest}>Start Test</button>
            )}

            {(method === "sequential" || method === "bayesian") &&
baselineConversion && minimumDetectableEffect && controlGroup && testGroup &&
conversionFrom && conversionTo && (
                <button onClick={startTest}>Start Test</button>
            )}
            {testStarted && (
                <label>Test successfully started</label>
```

```
            )}

        </div>
    );
}

export default App;
```

Додаток Б. Код файлу App.css

```
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}


.App {
  max-width: 500px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f8f8f8;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

h1 {
  text-align: center;
  margin-bottom: 20px;
```

```
  color: #333;
}

h2 {
  margin-bottom: 10px;
  color: #666;
}

label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
  color: #333;
}

input,
select,
button {
  width: 100%;
  padding: 8px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.slider-container {
  margin-bottom: 15px;
}

.slider-container span {
  display: inline-block;
  margin-left: 10px;
  font-weight: bold;
  color: #666;
}

.result {
  margin-top: 20px;
```

```
  padding: 10px;
  background-color: #eef5fd;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.parameter-container {
  display: flex;
  flex-direction: column;
  margin-bottom: 15px;
}

.parameter {
  display: flex;
  align-items: center;
  margin-bottom: 10px;
}

.parameter label {
  width: 200px;
  margin-right: 10px;
}

.parameter input {
  width: 150px;
  margin-right: 10px;
}

.parameter span {
  font-weight: bold;
  color: #666;
}
```

Додаток В. Код файлу index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
```

```
import './index.css';

ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById('root')
);
```

Додаток Г. Код файлу index.css

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
}

.container {
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
}

h1 {
  text-align: center;
  margin-bottom: 20px;
}

h2 {
  margin-bottom: 10px;
}

label {
  display: block;
  margin-bottom: 5px;
}

input[type="number"] {
```

```
  width: 100%;
  padding: 8px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

button {
  padding: 10px 20px;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #45a049;
}

p {
  margin-top: 20px;
  font-style: italic;
  text-align: center;
}

/* Media queries */
@media screen and (max-width: 480px) {
  .container {
    padding: 10px;
  }
}

@media screen and (max-width: 768px) {
  input[type="number"] {
    width: 80%;
  }
}
```

Додаток Ґ. Код файлу app.py

```
import numpy as np
from flask import Flask, request
from pymongo import MongoClient
from datetime import datetime
from flask_apscheduler import APScheduler
from calculations import check_verdict_for_fixed_base_horizon,
check_verdict_for_frequentist_interference, \
    check_verdict_for_sequential_sampling, check_verdict_for_bayesian_testing,
\
    calculate_sample_size_for_fixed_based_horizon
from mail_interaction import send_email

app = Flask(__name__)
app.config.from_pyfile('config.cfg')
client = MongoClient(app.config['MONGO_URI'])
db = client[app.config['MONGO_DB']]

mailApiKey = app.config['MAIL_API_KEY']
mail_from = app.config['FROM_EMAIL']
mail_to = app.config['TO_EMAIL']

scheduler = APScheduler()
scheduler.init_app(app)
scheduler.start()



@scheduler.task('interval', id='hourly_check', hours=1, misfire_grace_time=900)
def hourly_check():
    tests = db.tests_data.find({'finished': False})

    for test in tests:
        method = test['method']
        control_data = test['control_data']
        test_data = test['test_data']
        minimum_detectable_effect = test['minimum_detectable_effect']
        control_group_marker = test['control_group_marker']
```

```
        test_group_marker = test['test_group_marker']
        conversion_from = test['conversion_from']
        conversion_to = test['conversion_to']

        control_conversion = np.sum(control_data)
        control_size = len(control_data)
        control_conversion_rate = control_conversion / control_size
        test_conversion = np.sum(test_data)
        test_size = len(test_data)
        test_conversion_rate = test_conversion / test_size

        test_started = test['date_start']

        item_id =
f"{test_started}_{method}_{control_group_marker}_{test_group_marker}_{conversio
n_from}_{conversion_to}"

        subject = f"Test finished: {item_id}"
        message = f"Test method: {method}<br>" \
                  f"Test started: {test_started}<br><br>" \
                  f"Minimum Detectable Effect: {minimum_detectable_effect}<br>"
\
                  f"Control group marker: {control_group_marker}<br>" \
                  f"Test group marker: {test_group_marker}<br>" \
                  f"Conversion from: {conversion_from}<br>" \
                  f"Conversion to: {conversion_to}<br><br>" \
                  f"Control size: {control_size}<br>" \
                  f"Control conversion rate: {control_conversion_rate}<br>" \
                  f"Test size: {test_size}<br>" \
                  f"Test conversion rate: {test_conversion_rate}<br><br>"

        alpha = test['significance_level']
        power = test['statistical_power']
        beta = 1 - power

        if method == 'fixed':
```

```
            verdict = check_verdict_for_fixed_base_horizon(test['sample_size'],
control_size, test_size, control_conversion, test_conversion,
control_conversion_rate, test_conversion_rate, alpha)

            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'frequentist':
            verdict = check_verdict_for_frequentist_interference(alpha, beta,
power, minimum_detectable_effect)

            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'sequential':
            verdict = check_verdict_for_sequential_sampling(alpha, power,
minimum_detectable_effect, control_conversion, test_conversion, control_size,
test_size)

            if verdict:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)

        elif method == 'bayesian':
            verdict = check_verdict_for_bayesian_testing(test['epsilon'],
control_conversion, control_size, test_conversion, test_size)
```

```
            if verdict is not None:
                db.tests.update_one({'_id': test['_id']}, {'$set': {'finished':
True}})
                message += f"<b>Test finished with winning group:
{verdict}</b>"
                send_email(mailApiKey, mail_from, mail_to, subject, message)


@app.route('/sample_size_calculation', methods=['POST'])
def sample_size_calculation():
    method = str(request.json['method'])
    baseline_conversion = float(request.json['baselineConversion'])
    minimum_detectable_effect = float(request.json['minimumDetectableEffect'])
    statistical_power = float(request.json['statisticalPower'])
    significance_level = float(request.json['significanceLevel'])

    if method == 'fixed':
        sample_size =
calculate_sample_size_for_fixed_based_horizon(baseline_conversion,
minimum_detectable_effect, significance_level, statistical_power)
    else:
        sample_size = -1

    response = {'sampleSize': sample_size}
    return response


@app.route('/start_test', methods=['POST'])
def start_test():
    try:
        method = str(request.json['method'])
        baseline_conversion = float(request.json['baselineConversion'])
        minimum_detectable_effect =
float(request.json['minimumDetectableEffect'])
        control_group_marker = str(request.json['controlGroup'])
        test_group_marker = str(request.json['testGroup'])
        conversion_from = str(request.json['conversionFrom'])
        conversion_to = str(request.json['conversionTo'])
```

```
        date_start = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

        item_id =
f"{date_start}_{method}_{control_group_marker}_{test_group_marker}_{conversion_
from}_{conversion_to}"

        item = {
                '_id': item_id,
                'started': date_start,
                'method': method,
                'baseline_conversion': baseline_conversion,
                'minimum_detectable_effect': minimum_detectable_effect,
                'control_group_marker': control_group_marker,
                'test_group_marker': test_group_marker,
                'conversion_from': conversion_from,
                'conversion_to': conversion_to,
                'control_data': [],
                'test_data': [],
                'finished': False
            }

        if method =='fixed' or method == 'frequentist':
            item['statistical_power'] = float(request.json['statisticalPower'])
            item['significance_level'] =
float(request.json['significanceLevel'])
            if method == 'fixed':
                item['sample_size'] = float(request.json['sampleSize'])
        elif method == 'bayesian':
            item['epsilon'] = float(request.json['epsilon'])

        db.tests_data.insert_one(item)
        return {'started': True}
    except Exception as e:
        return {'started': False}


if __name__ == '__main__':
```

```
    app.run()
```

Додаток Д. Код файлу calculations.py

```
import numpy as np
import scipy.stats as stats


def calculate_sample_size_for_fixed_based_horizon(baseline_conversion,
minimum_detectable_effect, significance_level, statistical_power):
    p1 = baseline_conversion
    p2 = baseline_conversion * (1 + minimum_detectable_effect)
    pool_sd = np.sqrt((p1 * (1 - p1) + p2 * (1 - p2)) / 2)
    z_score = stats.norm.ppf(1 - significance_level / 2)
    sample_size = np.ceil(
        (2 * (z_score + stats.norm.ppf(statistical_power)) ** 2) /
minimum_detectable_effect ** 2 * pool_sd ** 2)
    return sample_size


def check_verdict_for_fixed_base_horizon(sample_size, control_size, test_size,
control_conversion, test_conversion, control_conversion_rate,
test_conversion_rate, alpha):
    pooled_conversion_rate = (control_conversion + test_conversion) /
(control_size + test_size)
    standard_error = np.sqrt(pooled_conversion_rate * (1 -
pooled_conversion_rate) * (1 / control_size + 1 / test_size))
    test_statistic = (test_conversion_rate - control_conversion_rate) /
standard_error

    p_value = 2 * (1 - stats.norm.cdf(np.abs(test_statistic)))

    if control_size >= sample_size and test_size >= sample_size:
        if p_value < alpha:
            if control_conversion_rate > 0:
                verdict = "B"
            else:
                verdict = "A"
```

```
        else:
            verdict = "no difference"
    else:
        verdict = None
    return verdict


def check_verdict_for_frequentist_interference(alpha, beta, power,
minimum_detectable_effect):
    z_score = stats.norm.ppf(1 - alpha / 2)
    p_value = stats.norm.cdf(z_score)

    if p_value < alpha:
        if power > 1 - beta:
            if minimum_detectable_effect > 0:
                verdict = "B"
            else:
                verdict = "A"
        else:
            verdict = None
    else:
        if power > 1 - beta:
            verdict = "no difference"
        else:
            verdict = None
    return verdict


def check_verdict_for_sequential_sampling(alpha, power,
minimum_detectable_effect, control_conversion, test_conversion, control_size,
test_size):
    pooled_conversion_rate = (control_conversion + test_conversion) /
(control_size + test_size)
    z_score = stats.norm.ppf(1 - alpha / 2)
    sample_size = np.ceil(
        (2 * (z_score + stats.norm.ppf(power)) ** 2) * (
                    pooled_conversion_rate * (1 - pooled_conversion_rate)) /
minimum_detectable_effect ** 2)
```

```
    if test_conversion - control_conversion > 2 * np.sqrt(sample_size):
        verdict = "B"
    elif test_conversion + control_conversion > sample_size:
        verdict = "no difference"
    else:
        verdict = None
    return verdict


def calculate_expected_loss(c, n):
    alpha = c + 1
    beta = n - c + 1
    expected_loss = 1 - stats.beta.mean(alpha, beta)
    return expected_loss


def choose_winner(control_conversion, control_size, variant_conversion,
variant_size, epsilon):
    loss_control = calculate_expected_loss(control_conversion, control_size)
    loss_variant = calculate_expected_loss(variant_conversion, variant_size)
    Q = set()

    if loss_control < epsilon:
        Q.add('A')
    if loss_variant < epsilon:
        Q.add('B')

    if len(Q) == 0:
        return None
    else:
        return Q.pop()


def check_verdict_for_bayesian_testing(epsilon, control_conversion,
control_size, test_conversion, test_size):
    return choose_winner(control_conversion, control_size, test_conversion,
test_size, epsilon)
```

Додаток Е. Код файлу mail_interaction.py

```
from sendgrid.helpers.mail import Mail
from sendgrid import SendGridAPIClient


def send_email(mailApiKey, mail_from, mail_to, subject, message_text):
    mess = Mail(
        from_email=mail_from,
        to_emails=mail_to,
        subject=subject,
        html_content=message_text
    )
    sg = SendGridAPIClient(mailApiKey)
    response = sg.send(mess)
```