Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

__Навчально-науковий інститут фізико-технічних та комп'ютерних наук___
(повна назва факультету, інституту)

_____кафедра кореляційної оптики_____
(повна назва кафедри, циклової комісії)

Курсова робота

з _____Інформатики_____
(назва дисципліни)

на тему:

Аналіз статистики використання мобільних мереж.

Студентки _1_ курсу _124_ групи

Спеціальність: «Електроні комунікації та радіотехніка»
(ОПП «Телекомунікації»)

Продан Д.В.

(підпис) (прізвище та ініціали)

Керівник:_____Рябий П. А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____ Оцінка: ECTS ____

Члени комісії: _____ _Іванський Д.І._____
(підпис) (прізвище та ініціали)

_____ _Городинська Н.В.__
(підпис) (прізвище та ініціали)

_____ _Олар О.В.___
(підпис) (прізвище та ініціали)

Чернівці – 2025

Реферат

Розробка комплексної системи для аналізу статистики використання мобільних мереж є ключовою метою цього проєкту. Система дозволить ефективно обробляти значні обсяги даних мережевого трафіку, виявляти важливі закономірності та наочно візуалізувати ключові показники. Для досягнення цього буде здійснено: генерацію великого масиву синтетичних даних трафіку; попередню обробку, очищення та аналіз цих даних; створення інтерактивної панелі для представлення результатів аналізу.

Обсяг роботи: сторінок - 68, джерел літератури - 5.

Список умовних позначень

1. CSV Comma-Separated Values (текстовий формат таблиць, розділених комами)
2. df DataFrame – таблична структура даних у pandas
3. n_rows Кількість рядків (записів) у датасеті
4. id Унікальний ідентифікатор з’єднання
5. start_time Час початку з’єднання
6. end_time Час завершення з’єднання
7. dur Duration – тривалість з’єднання в секундах
8. proto Network protocol – протокол передачі даних (TCP, UDP, ICMP, HTTP, HTTPS тощо)
9. service Сервіс/додаток (http, ftp, dns тощо)
10. state Стан з’єднання (EST, CON, REQ, RST тощо)
11. spkts Sent packets – кількість пакетів, відправлених джерелом
12. dpkts Received packets – кількість пакетів, отриманих приймачем
13. sbytes Sent bytes – обсяг даних (байтів), відправлених джерелом
14. dbytes Received bytes – обсяг даних (байтів), отриманих приймачем
15. rate Throughput – швидкість передачі даних (біт/с)
16. sttl / dttl Source/Destination TTL – Time To Live (максимальна “життєздатність” пакета)
17. swin / dwin Source/Destination Window Size – розмір вікна TCP у байтах
18. sloss / dloss Source/Destination Loss – кількість втрачених пакетів
19. sinpkt / dinpkt Source/Destination Inter-Packet Arrival Time – час між пакетами (середній)
20. sjit / djit Source/Destination Jitter – джиттер (варіація затримок)
21. tcprtt TCP Round-Trip Time – затримка «туди-назад» TCP-з’єднання
22. synack / ackdat Час встановлення TCP-з’єднання (SYN-ACK та ACK-DATA)
23. src_ip / dst_ip IP-адреса джерела та призначення
24. src_port / dst_port Порт джерела та порт призначення

25. src_country / dst_country Країна-джерело та країна-призначення трафіку
26. DDoS Distributed Denial-of-Service – розподілена атака відмови в обслуговуванні
27. TTL Time To Live – “час життя” пакета в мережі
28. TCP Transmission Control Protocol
29. UDP User Datagram Protocol
30. HTTP/HTTPS HyperText Transfer Protocol (безпечний через SSL/TLS)
31. ICMP Internet Control Message Protocol
32. KPI Key Performance Indicator – ключовий показник ефективності

Вступ

Мета роботи:

Створення зручного інструменту для аналізу даних мобільних мереж, який допоможе операторам краще розуміти стан мережі, виявляти технічні неполадки, слідкувати за навантаженням, запобігати шахрайству та підвищувати рівень безпеки і якості обслуговування.

Сценарій застосування:

У реальних умовах оператори мобільного зв’язку щодня стикаються з великими обсягами даних, які генерує мережа: дзвінки, передача даних, повідомлення, підключення до вишок тощо. За допомогою запропонованого інструменту вони зможуть:

- відстежувати трафік і навантаження на мережу в різних регіонах;
- аналізувати довготривалі тенденції використання для планування розширення мережі;
- отримувати візуальні звіти для швидкого прийняття рішень.

Такий підхід дозволяє ефективніше управляти мережею, економити ресурси та покращувати досвід користувачів.

Зміст

1. Постановка задачі

Метою цього проєкту є розробка зручної та доступної системи для аналізу трафіку мобільних мереж. Ключовими користувачами цієї системи стануть оператори мобільного зв'язку, яким вона надасть можливість глибше розуміти функціонування їхніх мереж, виявляти проблемні місця та приймати рішення для оптимізації роботи.

Для успішного досягнення поставленої мети передбачено виконання кількох важливих етапів:

1.1 Формування великих масивів штучних даних

- Розробити механізм для генерації даних, що симулюватимуть активність користувачів у мобільній мережі.
- Врахувати ключові параметри, такі як типи трафіку (HTTP, FTP та інші), обсяг переданих даних, кількість встановлених з'єднань, тривалість сесій та інше.
- Розглянути можливість застосування вже наявних наборів даних, знайдених в інтернеті, зокрема, з платформи Kaggle. [1]

1.2 Підготовка та опрацювання даних

- Виконати очищення даних від помилок, дублювань та небажаних шумів.
- Підготувати дані до проведення подальшого аналізу.

1.3Візуалізація результатів аналізу

- Створити просту та інтуїтивно зрозумілу панель моніторингу, на якій будуть відображені ключові показники.
- Забезпечити наявність графіків, таблиць, діаграм та карт для зручного сприйняття інформації.
- Зробити панель інтерактивною: передбачити функціонал фільтрації, сортування та налаштування представлення даних.

- Надати користувачу можливість визначати та налаштовувати набір необхідних метрик.

2. Аналіз вимог до програми

Технологічна постановка задачі

2.1 Модуль генерації даних

- **Мета**: Розробити скрипт dataset.py для створення контрольованих синтетичних даних мережевого трафіку.
- **Основні вимоги**:
  - Відтворюваність результатів (фіксація початкових значень генератора випадкових чисел).
  - Обсяг вибірки — не менше 230,000 записів.
  - Наявність понад 30 параметрів для кожного запису.
  - Наявність маркування трафіку як **звичайний** чи **потенційно шкідливий**.

2.2 Модуль обробки та аналізу даних

- **Мета**: Розробити скрипти data_loader.py та data_cleaner.py для завантаження й очищення даних.
- **Основні функції**:
  - Завантаження даних із CSV-файлів.
  - Обробка числових та категоріальних параметрів.
  - Фільтрація даних за протоколами, сервісами, часовими інтервалами.
  - Розрахунок статистичних метрик мережевої активності.

2.3 Модуль візуалізації та презентації даних

- **Мета**: Створити скрипт dashboard.py для інтерактивної візуалізації даних.
- **Основні функції**:
  - Побудова інтерактивної панелі з фільтрами.
  - Використання різноманітних типів графіків: гістограми, точкові діаграми, теплові карти.

- Виявлення та відображення аномального трафіку.

Функціональні вимоги

2.5 Генерація даних

- Створення синтетичних наборів даних із заданими параметрами.
- Підтримка кількох мережевих протоколів (TCP, UDP, HTTP, HTTPS тощо). [2]
- Генерація часових міток для всіх подій.
- Маркування типу трафіку (звичайний/аномальний).

2.6 Обробка та аналіз даних

- Завантаження даних із CSV-файлів.
- Фільтрація за протоколами, сервісами, часовими інтервалами.
- Розрахунок агрегованих метрик, таких як середня тривалість сесії, обсяг переданих даних.
- Виявлення статистичних закономірностей і аномалій.

2.7 Візуалізація

- Інтерактивна панель моніторингу з фільтрами.
- Побудова гістограм, стовпчикових і точкових діаграм.
- Побудова теплової карти кореляцій між числовими параметрами.
- Відображення основних метрик активності для різних категорій трафіку.

3. Опис алгоритму

Загальна структура проекту

Проект представляє собою інтерактивну аналітичну панель для візуалізації та аналізу мережевого трафіку, реалізовану на Python з використанням бібліотеки Streamlit.

3.1 Основні компоненти системи

1. Створення тестових даних (dataset.py)
    - Програма створює 230,000 прикладів мережевих з'єднань
    - Для кожного з'єднання вказується тип протоколу (наприклад, HTTP, FTP), сервіс та статус [3]
    - Для кожного з'єднання також вказуються показники: як довго тривало, скільки даних передано
    - Всі ці дані зберігаються у файл формату CSV (схожий на Excel-таблицю)
2. Читання та підготовка даних
    - Програма може працювати як з реальними даними, так і з тестовими
    - Модуль data_loader.py завантажує дані з файлів
    - Модуль data_cleaner.py перевіряє дані та виправляє можливі помилки
    - Файл config.py містить налаштування для роботи з числовими даними
3. Інтерактивний інтерфейс (dashboard.py)
    - Створює веб-сторінку з графіками та можливістю взаємодії
    - Дозволяє перемикатися між реальними та тестовими даними(більш розширений варіант даних)
    - Надає фільтри для вибору конкретних протоколів або сервісів
    - Показує важливі показники (кількість з'єднань, середня тривалість)
    - Містить шість різних вкладок з різними типами графіків

3.2 Алгоритм роботи проекту

Запуск програми:

- Налаштовується веб-інтерфейс
- Завантажуються дані (користувач може вибрати реальні або тестові)

Фільтрація:

- Користувач може вибрати, які типи з'єднань його цікавлять
- Система відображає тільки ті дані, які відповідають вибраним фільтрам

Показ основних цифр:

- Розраховуються загальні показники (кількість з'єднань, об'єм даних)
- Ці показники відображаються у вигляді зручних лічильників

Створення графіків:

- Розподіли: Графіки, що показують розподіл значень (наприклад, скільки з'єднань якої тривалості)
- Кореляції: Теплова карта, яка показує, як пов'язані різні показники між собою
- Протоколи: Кругові та стовпчикові діаграми про різні типи з'єднань
- Загальний огляд: Чотири найважливіші графіки на одному екрані
- Часовий аналіз: Графіки активності за годинами доби та днями тижня
- Карта світу: Інтерактивна карта, яка показує звідки і куди йде трафік

Які технології використовуються?

- Python: Основна мова програмування
- Pandas: Інструмент для роботи з таблицями даних
- Numpy: Бібліотека для математичних розрахунків
- Streamlit: Інструмент для створення веб-інтерфейсу без знання веб-розробки. [4]

- Plotly: Бібліотека для створення інтерактивних графіків і карт, на які можна наводити мишкою та отримувати додаткову інформацію

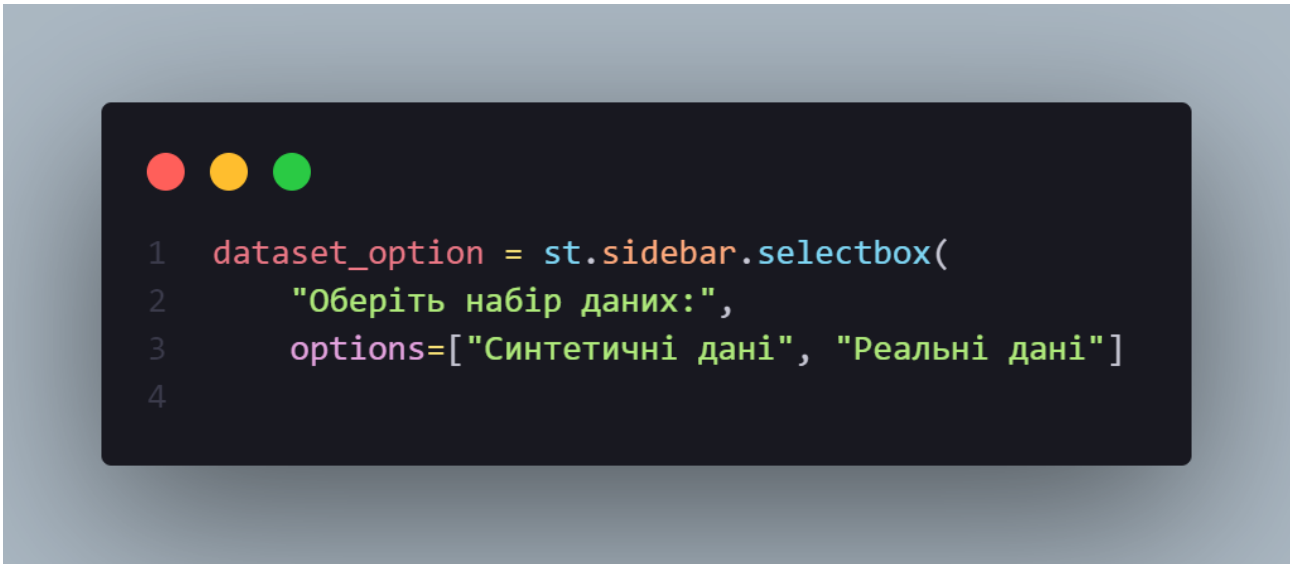
3.3 Ризики застосування синтетичних даних

1) Нереалістичність розподілів
   - Синтезовані дані можуть не відтворювати справжні статистичні тренди.
   - Штучні розподіли здатні створювати оманливе враження про властивості трафіку. [5]
   - Використання синтетики може зумовити помилкові результати аналізу.
2) Відсутність реальних аномалій
   - Сгенеровані дані здебільшого не містять нетипових ситуацій, що властиві реальному трафіку.
   - Це здатне призвести до неготовності реагувати на справжні виклики.
3) Спрощеність моделей взаємодії
   - Взаємозв'язки між різними параметрами у синтезованих даних можуть бути надто спрощеними.
   - Кореляції можуть мати штучний характер або взагалі не існувати там, де вони важливі.
4) Обмежена географічна репрезентація
   - У проєкті використовується лише 8 країн, тоді як реальний трафік є глобальним явищем.
   - Це звужує можливості для аналізу географічних особливостей мережевого трафіку.
5) Відсутність сезонності та специфічних патернів
   - Реальний трафік часто має сезонні коливання та унікальні особливості, яких бракує синтетичним даним.
   - Це ускладнює прогнозування та виявлення потенційних проблем.

3.4 Фільтри в системі аналізу мережевого трафіку

1. Основні фільтри (бічна панель)

Вибір набору даних



```
dataset_option = st.sidebar.selectbox(
    "Оберіть набір даних:",
    options=["Синтетичні дані", "Реальні дані"]

```

Рис. 1 Приклад в коді

- Дозволяє перемикатися між синтетичним датасетом та реальними даними
- Повністю змінює джерело даних для аналізу

Фільтри мережевого трафіку

```

protocols = ['Всі'] + sorted(df['proto'].unique().tolist())
selected_protocol = st.sidebar.selectbox("Протокол:", protocols)

services = ['Всі'] + sorted(df['service'].unique().tolist())
selected_service = st.sidebar.selectbox("Сервіс:", services)

states = ['Всі'] + sorted(df['state'].unique().tolist())
selected_state = st.sidebar.selectbox("Стан:", states)

```

Рис. 2 Приклад в коді.

- Кожен фільтр обмежує відображені дані відповідно до вибраного значення

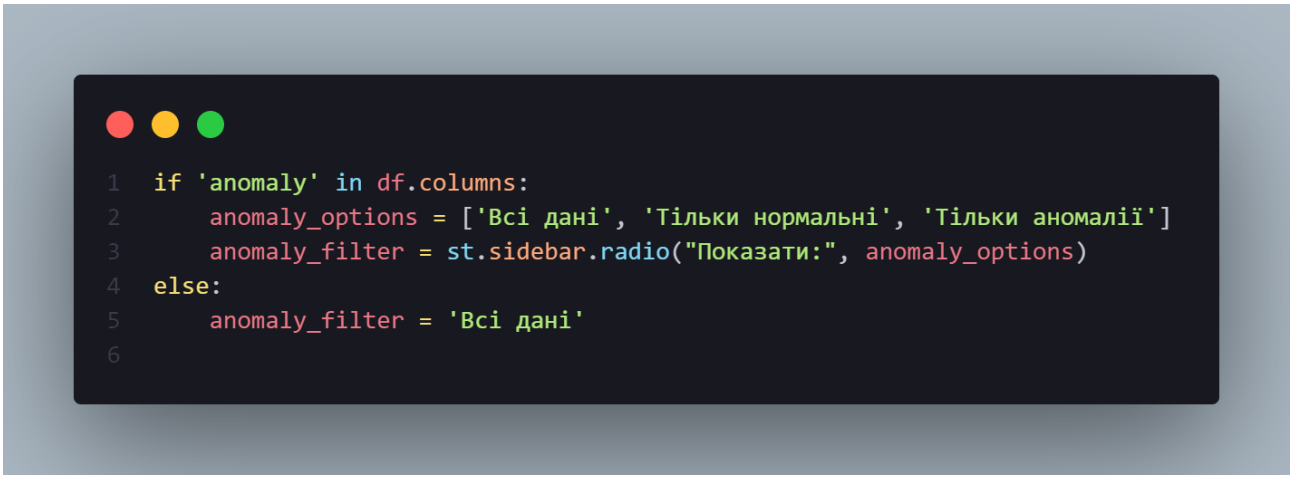
- Логіка застосування: [filtered_df = filtered_df[filtered_df['proto'] == selected_protocol]](http://vscodecontentref/0)
- Значення "Всі" пропускає всі дані для даного параметра

Часовий фільтр

Рис. 3 Приклад в коді

- Фільтрує дані в межах вибраного діапазону дат
- Логіка: date_mask = (df['date'] >= start_date) & (df['date'] <= end_date)
- За замовчуванням показує весь доступний діапазон часу

Фільтр аномалій



```
if 'anomaly' in df.columns:
    anomaly_options = ['Всі дані', 'Тільки нормальні', 'Тільки аномалії']
    anomaly_filter = st.sidebar.radio("Показати:", anomaly_options)
else:
    anomaly_filter = 'Всі дані'

```

Рис. 4 Приклад в коді

- Дозволяє фокусуватись на нормальному трафіку або лише на аномаліях

3.5 Процес виявлення аномалій

Виконує три типи алгоритмів виявлення:

DDoS атаки

- Виявляє з'єднання з надзвичайно високою швидкістю пакетів (>1000 пакетів/с)
- Відзначає короткі з'єднання (<0.1 с) з високою швидкістю

Помилкові конфігурації

- Неправильні TTL значення (≤2 або ≥254)
- Нетипові розміри вікна TCP (≤3 або ≥65534)
- Аномальне співвідношення пакетів до байтів (>100 пакетів зі середнім розміром <10 байт)

Нестандартні порти

- Використовує мапу відповідності сервісів та стандартних портів
- Виявляє з'єднання, де сервіс використовує нестандартний порт

Після виявлення

Система показує:

- Загальну кількість виявлених аномалій
- Розподіл за типами аномалій
- Порівняння характеристик нормального та аномального трафіку
- Приклади аномальних з'єднань

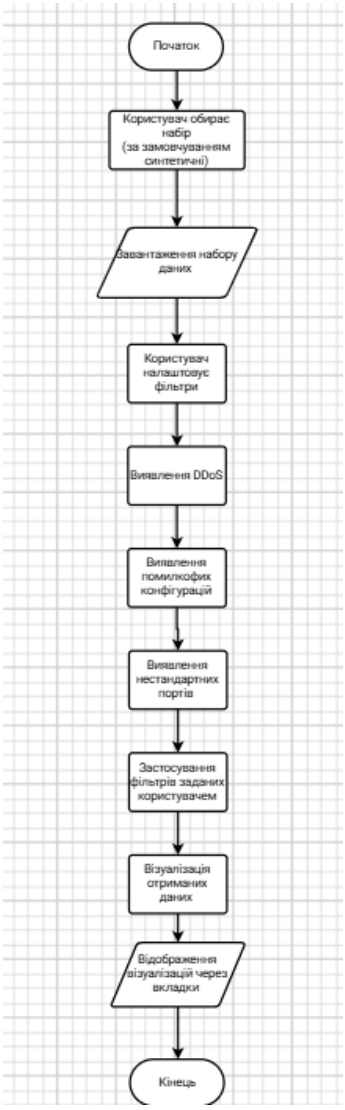


Рис.5 Реалізована блок-схема проекту

4. Технічні виклики та їх подолання

У проекті реалізовано комплексний підхід до обробки екстремальних значень, що дозволяє коректно відображати дані та запобігати спотворенню візуалізації:

- Обмеження діапазону відображення

```
fig.add_trace(go.Box(y=df[df['anomaly'] == 0]['dur'].clip(upper=20),
                     name='Нормальний трафік',
                     marker_color='blue'))
```

- Логарифмічні шкали для показників з великим діапазоном значень:

```
fig = px.scatter(filtered_df, x="spkts", y="sbytes",
                 size="dur", color="proto", hover_name="service",
                 log_x=True, log_y=True,
                 labels={"spkts": "Пакети", "sbytes": "Байти", "dur": "Тривалість"})
```

- Нормалізація значень для порівнянних графіків:

```
normal_data['pkt_to_byte_ratio'] = normal_data['spkts'] / (normal_data['sbytes'] + 1)
```

Для забезпечення цілісності аналізу впроваджено багаторівневу систему обробки відсутніх даних

- Заповнення нулями для часових рядів:

```
rate_df = rate_df.fillna(0).sort_values('time')
```

- Умовна обробка при агрегації даних для запобігання помилок

```
anomaly_percent = (anomaly_counts.get(1, 0) / total_connections) * 100 if total_connections > 0 else 0
```

- Безпечне виконання операцій з використанням захисту від ділення на нуль

```
anomaly_data['pkt_to_byte_ratio'] = anomaly_data['spkts'] / (anomaly_data['sbytes'] + 1)
```

Механізми валідації вхідних даних

Створено багатоетапну систему валідації для забезпечення коректності аналізу:

- Попередня обробка через функцію clean_data():

```
df = clean_data(df, NUMERIC_COLUMNS)
```

- Перевірка формату дат при створенні часових фільтрів

```
if 'start_time' in df.columns:
    df['start_time'] = pd.to_datetime(df['start_time'])
    df['hour'] = df['start_time'].dt.hour
    df['date'] = df['start_time'].dt.date
```

- Валідація значень для фільтрів при формуванні запитів

```
if selected_protocol != 'Всі':
    filtered_df = filtered_df[filtered_df['proto'] == selected_protocol]
```

- Обробка граничних випадків через систему масок у pandas

```
ddos_mask = (df['spkts'] / df['dur'] > ddos_threshold_pkt_rate) & (df['dur'] < 0.1)
```

4.1 Оптимізація візуалізації

Відображення великих обсягів даних на графіках

Для ефективної візуалізації великих масивів даних впроваджено:

- Агрегація даних перед візуалізацією

```
hourly_traffic = filtered_df.groupby('hour')['sbytes'].sum().reset_index()
day_hour_traffic = df.groupby(['day_of_week', 'hour'])['sbytes'].sum().reset_index()
```

- Вибіркове відображення найбільш інформативних елементів

```
service_counts = filtered_df['service'].value_counts().head(10)  # Тільки топ-10
```

- Розподіл даних за вкладками для зменшення одночасного навантаження

tab1, tab2, tab3, tab4, tab5, tab6, tab7 = st.tabs(["Розподіли", "Кореляції", ...])

Адаптивний інтерфейс для різних розмірів екрану

Інтерфейс пристосований до різних умов відображення

Широкий макет за замовчуванням

st.set_page_config(layout="wide", page_title="Аналіз мережевого трафіку")

Динамічне розташування елементів з використанням сітки колонок

col1, col2, col3, col4 = st.columns(4) # Рівномірне розташування метрик

Адаптивна ширина графіків для заповнення доступного простору

st.plotly_chart(fig, use_container_width=True)

Компактні елементи керування у бічній панелі

st.sidebar.selectbox("Протокол:", protocols)

Горизонтальне розташування елементів керування для економії вертикального простору

traffic_direction = st.radio("Напрямок трафіку:", options=["Джерело", "Призначення"], horizontal=True)

4.2 Система кешування для прискорення завантаження даних

Для підвищення продуктивності при роботі з даними

- Одноразове завантаження даних на початку сеансу

with st.spinner("Завантаження даних..."):

    df_real = load_dataset('real')

    df_synthetic = load_dataset()

- Повторне використання відфільтрованих даних замість створення нових копій

```
filtered_df = df.copy()  # Створення копії лише один раз перед застосуванням фільтрів
```

- Обчислення проміжних результатів для багаторазового використання

```
# Розрахунок виконується один раз і результат використовується у різних графіках
anomaly_counts = df['anomaly'].value_counts()
```

- Поступова агрегація даних для зменшення обсягу операцій

```
# Агрегація спочатку на рівні країн, потім інші операції
country_traffic = filtered_df.groupby(country_col).agg(
    total_bytes=pd.NamedAgg(column='sbytes', aggfunc='sum'),
    count=pd.NamedAgg(column='id', aggfunc='count')
)
```

Ці технічні рішення забезпечують стабільну роботу системи, підвищують точність аналізу та надають користувачу зручний інструментарій для візуальної аналітики навіть при роботі з надзвичайно великими або складними наборами даних.

5. Порівняння з аналогічними рішеннями

Функціонал	Даний проект	Wireshark	Kibana	Splunk
Інтуїтивний інтерфейс	✅	❌	✅	✅
Географічна візуалізація	✅	❌	✅	✅
Виявлення аномалій	✅ (базове)	❌	✅	✅
Аналіз протоколів	✅ (обмежений)	✅ (детальний)	✅	✅
Масштабованість	❌	❌	✅	✅

Табл.1 Порівняння з аналогами

6. Приклади роботи

Основні частини коду для демонстрації проекту

6.1 Генерація даних (dataset.py)

```
# Визначення категоріальних значень
protocols = ['TCP', 'UDP', 'ICMP', 'HTTP', 'HTTPS', 'DNS', 'FTP', 'SMTP', 'SSH']
services = ['http', 'https', 'dns', 'ftp', 'ssh', 'smtp', 'pop3', 'imap', 'telnet',
            'ntp', 'dhcp', 'rdp', 'vnc', '-', 'other']
states = ['FIN', 'CON', 'INT', 'RST', 'ACC', 'REQ', 'CLO', 'EST', '-']
# Створення DataFrame
df = pd.DataFrame({
    'proto': np.random.choice(protocols, size=n_rows, p=[0.4, 0.3, 0.1, 0.05, 0.05, 0.04, 0.02, 0.02, 0.02]),
    'service': np.random.choice(services, size=n_rows),
    'dur': np.random.exponential(10, n_rows),  # Тривалість у секундах
    'spkts': np.random.randint(1, 1000, n_rows),  # Відправлені пакети
    'dpkts': np.random.randint(1, 1000, n_rows),  # Отримані пакети
    'sbytes': np.random.randint(100, 10000000, n_rows),  # Відправлені байти
    # ... інші поля
})
```

Весь код файлу наведено у Додатку А

6.2 Основний інтерфейс панелі (dashboard.py)

```
def main():
    st.set_page_config(layout="wide", page_title="Аналіз мережевого трафіку")
```

```
st.title("Інтерактивна панель аналізу мережевого трафіку")
with st.spinner("Завантаження даних..."):
    df = load_data()
    df = clean_data(df, NUMERIC_COLUMNS)
st.sidebar.title("Фільтри")
selected_protocols = st.sidebar.multiselect(
    "Оберіть протоколи:",
    options=df['proto'].unique(),
    default=df['proto'].value_counts().nlargest(3).index.tolist()
)

selected_services = st.sidebar.multiselect(
    "Оберіть сервіси:",
    options=df['service'].unique(),
    default=df['service'].value_counts().nlargest(3).index.tolist()
)
```

Весь код файлу наведено у Додатку Б

А також код очищення та завантаження даних буде в Додатку В

6.3 Відображення ключових метрик

```
st.header("Ключові метрики")
col1, col2, col3, col4 = st.columns(4)
col1.metric("Кількість сесій", f"{len(filtered_df):,}")
col2.metric("Середня тривалість", f"{filtered_df['dur'].mean():.2f} c")
```

```
col3.metric("Загальний обсяг даних", f"{filtered_df['sbytes'].sum():,} байтів")
col4.metric("Найпоширеніший
протокол",filtered_df['proto'].value_counts().index[0])
```

6.4 Складна візуалізація з декількома графіками

```
with tab4:
    fig = make_subplots(
        rows=2, cols=2,
        subplot_titles=("Розподіл даних", "Топ сервіси",
                        "Залежність пакетів", "Тривалість за протоколами")
    )
    fig.add_trace(
        go.Histogram(x=filtered_df['sbytes'], name="Відправлені байти"),
        row=1, col=1
    )
    # ... додаткові графіки
    fig.update_layout(height=800)
    st.plotly_chart(fig, use_container_width=True)
```

6.5 Приклад структури CSV файлу для проекту аналізу мережевого трафіку

Нижче наведено приклад того, як виглядає CSV файл із даними про мережевий трафік, який використовується в проекті:

```
id,start_time,proto,service,state,dur,sbytes,dbytes,spkts,dpkts,src_country,dst_country
1,2023-05-08 10:15:23,TCP,HTTP,ESTABLISHED,2.35,1250,350,15,10,Україна,США
2,2023-05-08 10:16:45,UDP,DNS,ESTABLISHED,0.45,120,240,2,1,Україна,Німеччина
3,2023-05-08 10:17:12,TCP,HTTPS,ESTABLISHED,5.64,45000,2300,120,85,США,Україна
4,2023-05-08 10:18:30,TCP,SSH,ESTABLISHED,120.5,15240,8560,340,220,Польща,Україна
5,2023-05-08 10:20:11,UDP,NTP,ESTABLISHED,0.12,76,76,1,1,Україна,Франція
6,2023-05-08 10:21:05,TCP,FTP,ESTABLISHED,45.8,256000,1200,540,45,Китай,Україна
7,2023-05-08 10:23:30,TCP,SMTP,ESTABLISHED,1.25,4500,350,25,12,Україна,Велика Британія
8,2023-05-08 10:25:42,UDP,SNMP,ESTABLISHED,0.35,560,120,4,3,Канада,Україна
```

Рис.6 Приклад структури даних датасету

- id: Унікальний ідентифікатор запису
- start_time: Час початку з'єднання
- proto: Протокол передачі даних (TCP, UDP тощо)
- service: Тип сервісу (HTTP, DNS, FTP тощо)
- state: Стан з'єднання (ESTABLISHED, CLOSED, TIME_WAIT тощо)
- dur: Тривалість з'єднання в секундах
- sbytes: Кількість байтів, відправлених від джерела
- dbytes: Кількість байтів, відправлених від отримувача
- spkts: Кількість пакетів, відправлених від джерела
- dpkts: Кількість пакетів, відправлених від отримувача
- src_country: Країна-джерело трафіку
- dst_country: Країна-призначення трафіку

7. Візуалізація даних

Головна сторінка

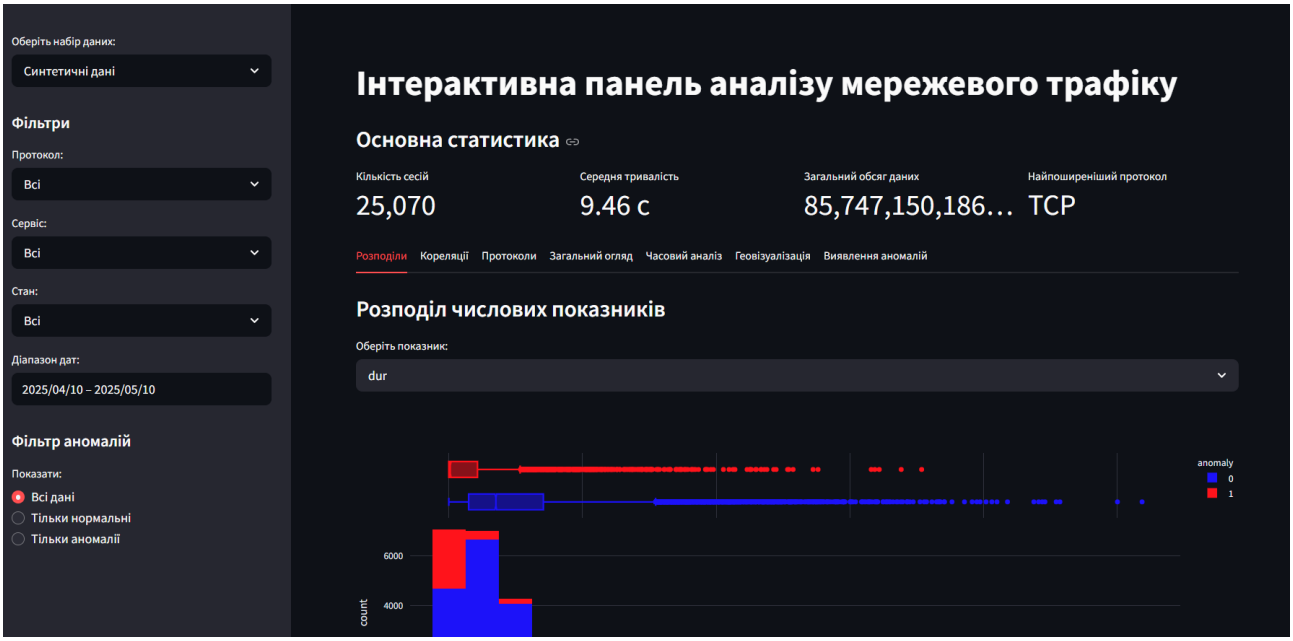


Рис.7 Відображення інтерактивного дашборду

Фільтри:

- Протокол: Дозволяє фільтрувати дані за певним мережевим протоколом
- Сервіс: Можливість фільтрувати за певним сервісом
- Станс: Ще один фільтр, схоже на стан або статус
- Діапазон дат: Дозволяє вказати період, за який потрібно аналізувати дані

Вкладка протоколи

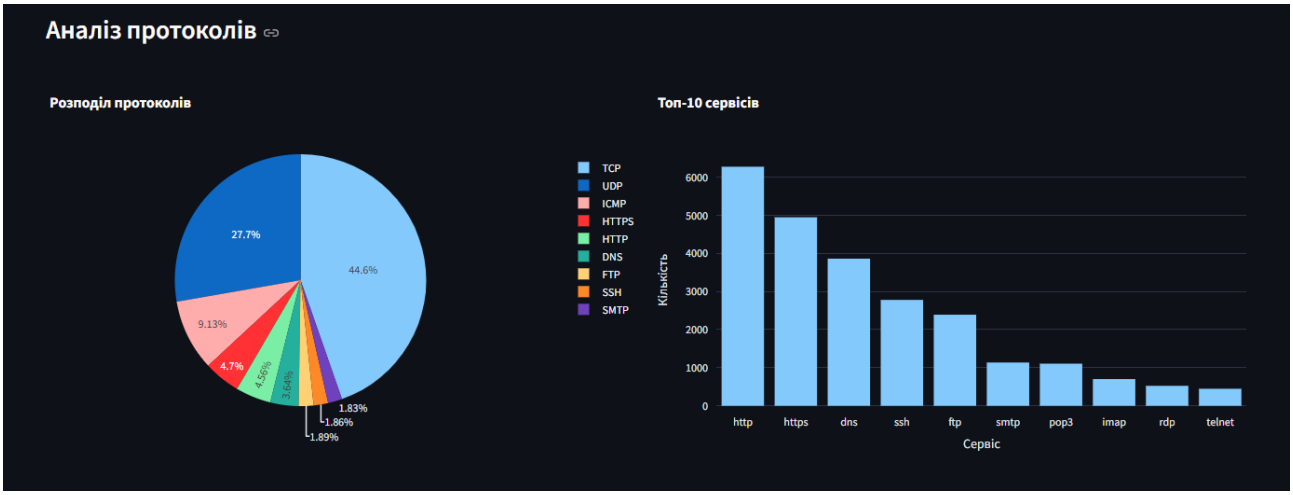


Рис.8 Відображення графіків аналізу протоколів

Вкладка кореляція

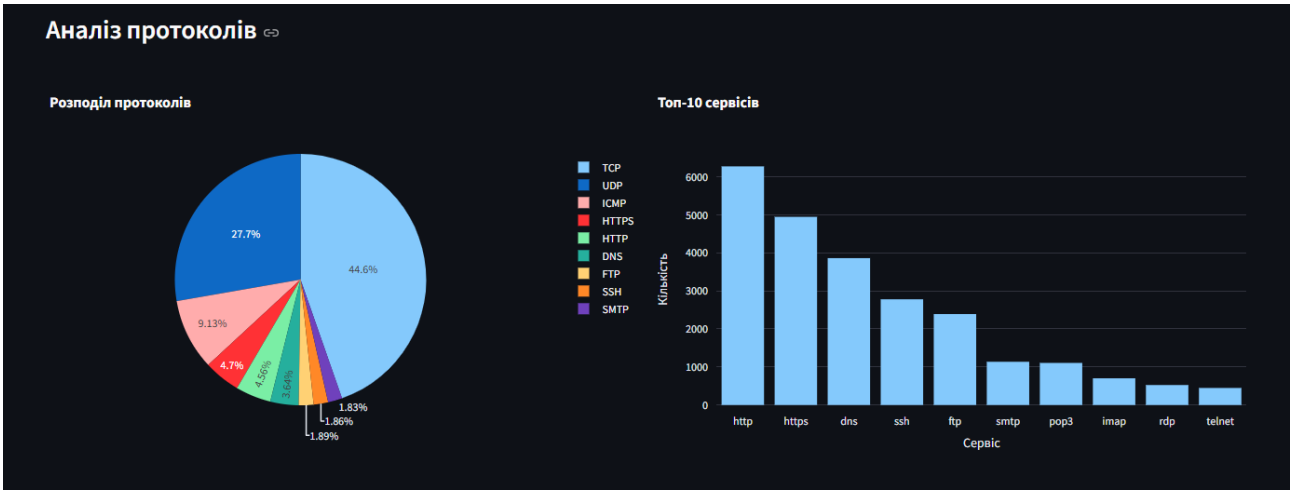


Рис.9 Кореляційна матриця

Вкладка загальний огляд трафіку

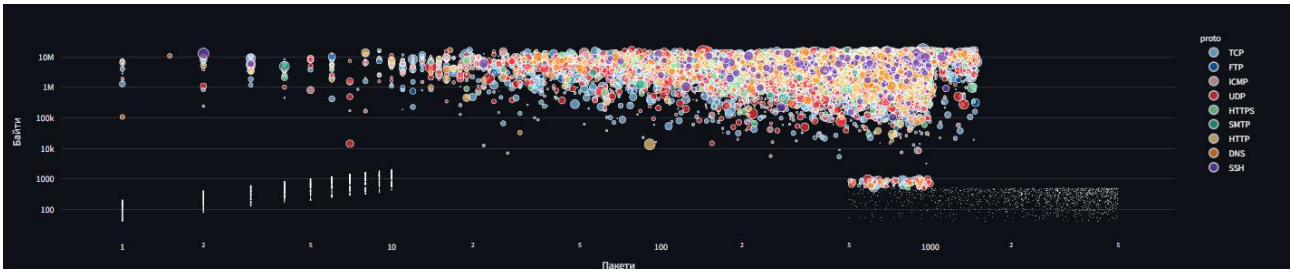


Рис.10Співвідношення пакетів та байт за протоколами

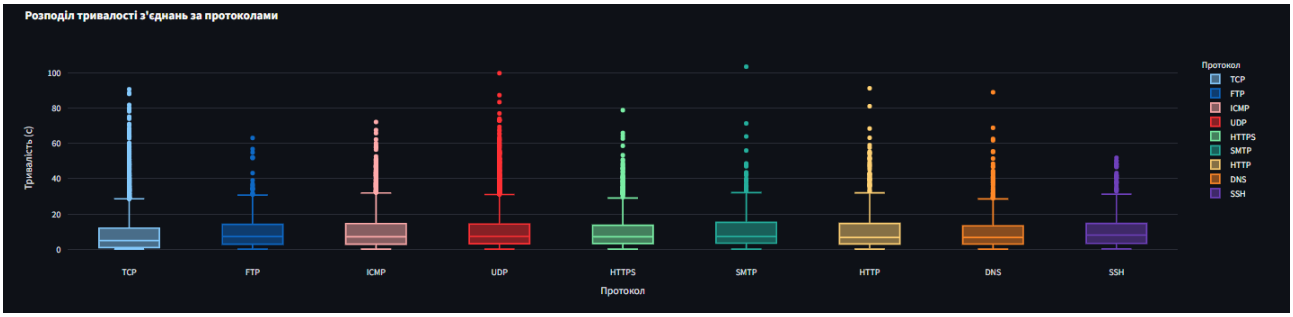


Рис.11 Розподіл тривалості зєднань за протоколами

Вкладка часовий розподіл

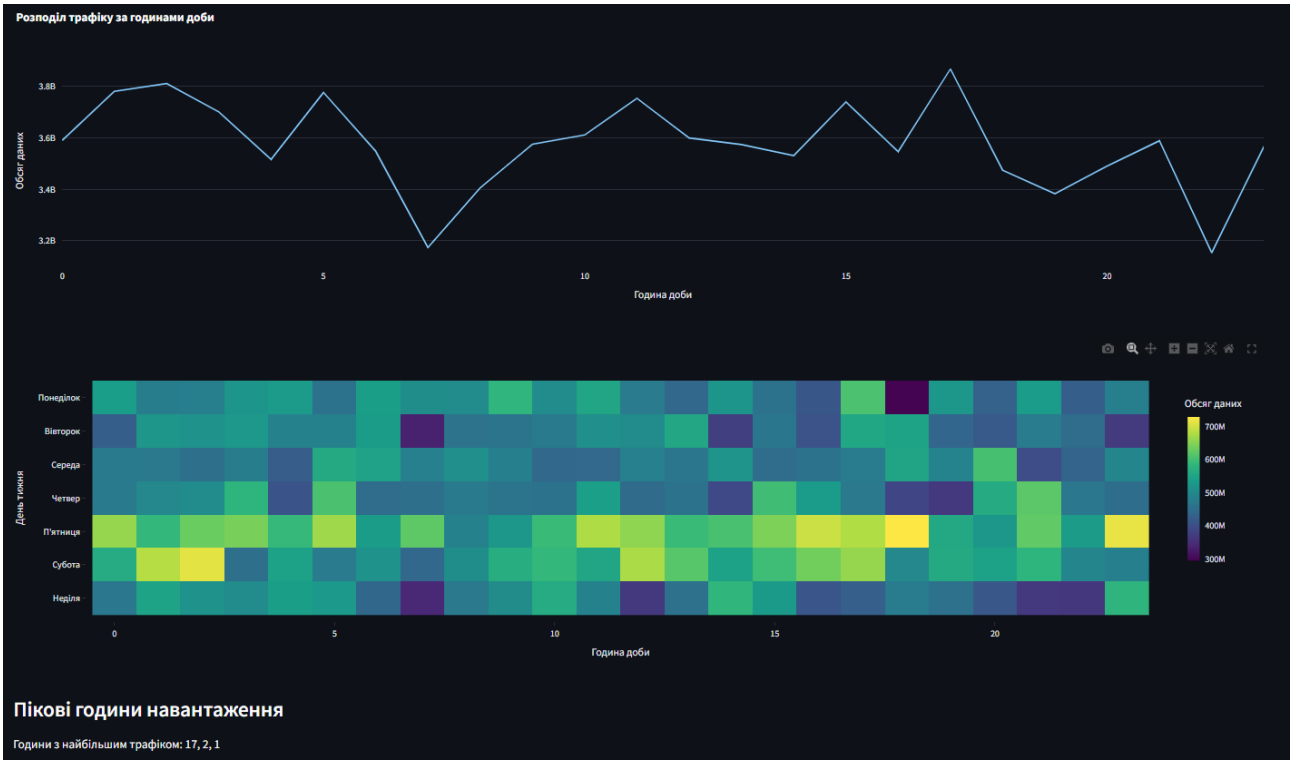


Рис.12 Розподіл трафіку за годинами доби

Також відображуються пікові години навантаження

Вкладка геовізуалізація

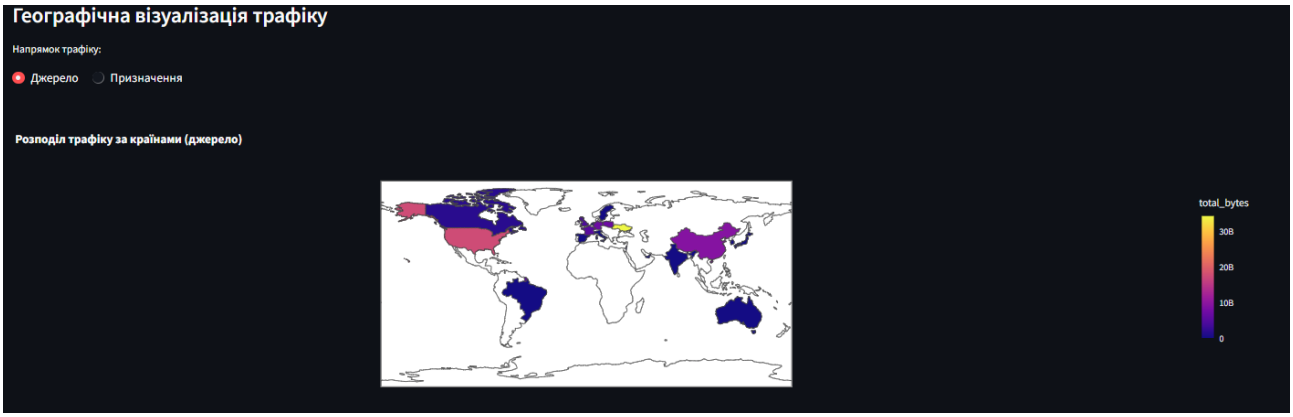
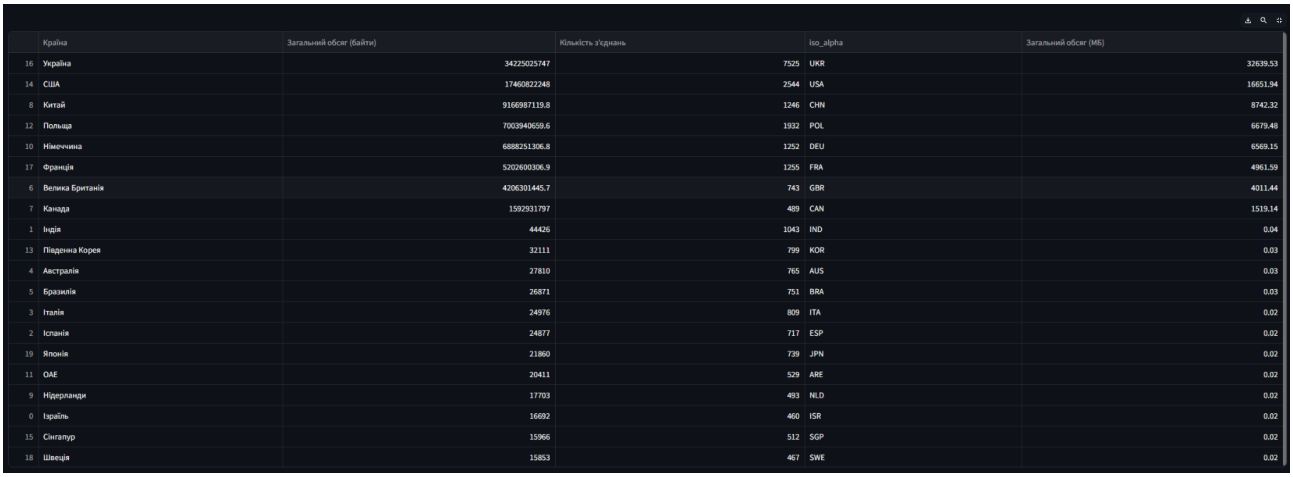


Рис. 13 Географічна візуалізація трафіку

Можливий вибір двох категорій “Джерело” “Призначення” трафіку



	Країна	Загальний обсяг (байти)	Кількість з'єднань	iso_alpha	Загальний обсяг (МБ)
16	Україна	34225025747	7525	UKR	32639.53
14	США	17460822248	2544	USA	16651.94
8	Китай	9166987119.8	1246	CHN	8742.32
12	Польща	7003940659.6	1932	POL	6679.48
10	Німеччина	6888251306.8	1252	DEU	6569.15
17	Франція	5202600306.9	1255	FRA	4961.59
6	Велика Британія	4206301445.7	743	GBR	4011.44
7	Канада	1592931797	489	CAN	1519.14
1	Індія	44426	1043	IND	0.04
13	Південна Корея	32111	799	KOR	0.03
4	Австралія	27810	765	AUS	0.03
5	Бразилія	26871	751	BRA	0.03
3	Італія	24976	809	ITA	0.02
2	Іспанія	24877	717	ESP	0.02
19	Японія	21860	739	JPN	0.02
11	ОАЕ	20411	529	ARE	0.02
9	Нідерланди	17703	493	NLD	0.02
0	Ізраїль	16692	460	ISR	0.02
15	Сінгапур	15966	512	SGP	0.02
18	Швеція	15853	467	SWE	0.02

Рис.14 Таблиця Деталі трафіку за країнами

Також за необхідності таблицю можна завантажити у форматі .csv

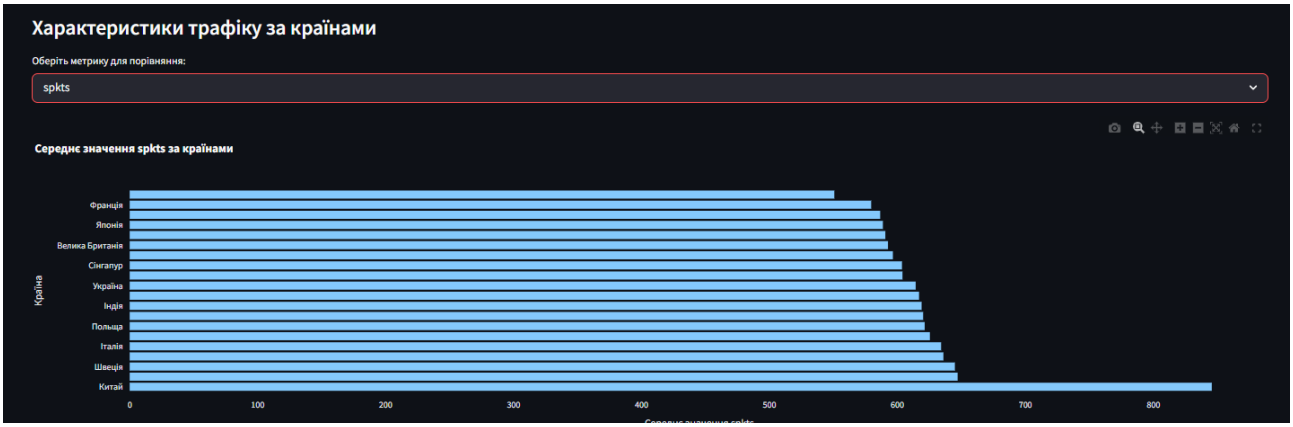


Рис.15 Характеристика трафіку за країнами

Вкладка виявлення аномалій

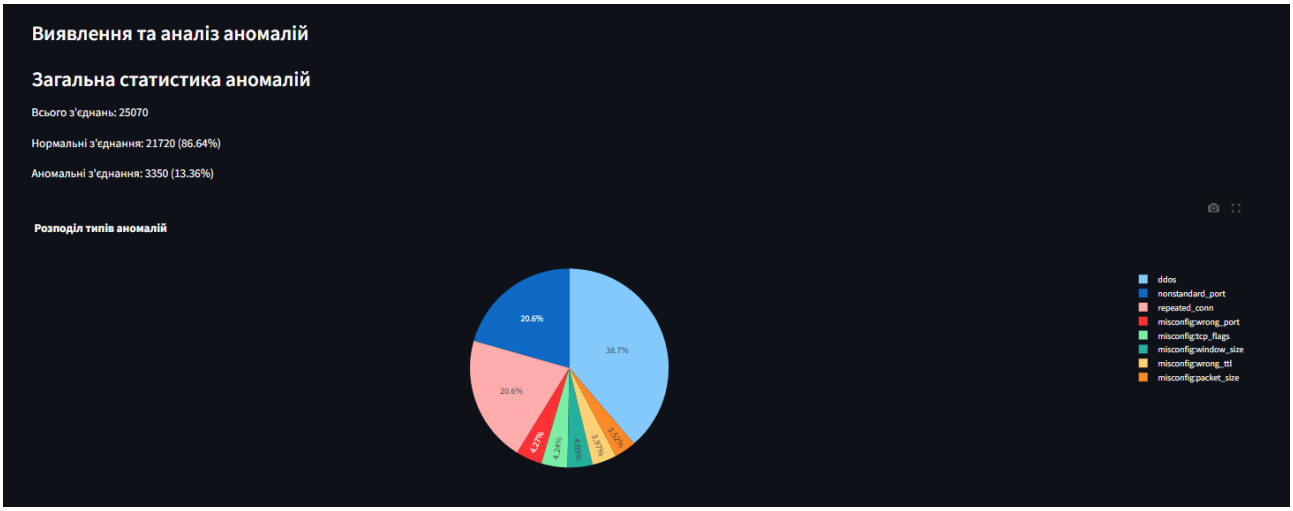


Рис.16 Загальна статистика

Рис.17 Детальна статистика

Висновки

У цій роботі створено інтуїтивно зрозумілий інструмент для аналізу мобільного мережевого трафіку, який спрощує збір та обробку даних, зменшуючи ручну участь. Система автоматично імпортує великі обсяги пакетних даних, виконує їх базову перевірку на помилки та пропонує готові звіти через зручний інтерфейс.

Інтерактивна панель дає змогу:

- **Фільтрувати дані** за протоколами (наприклад, HTTP, DNS), окремими сервісами та часовими інтервалами.
- **Візуалізувати ключові метрики** — обсяги трафіку, кількість з'єднань, середню затримку — у вигляді графіків та таблиць.
- **Виявляти аномалії**, такі як надто високий рівень пакетів чи підозрілі з'єднання, що допомагає у запобіганні шахрайству та витокам даних.

Завдяки такому підходу оператори та адміністратори мережі можуть оперативно:

- **Оцінювати навантаження** на різні вузли та визначати "зони" пікового трафіку.
- **Планувати розширення** або модернізацію обладнання на основі реальних показників.
- **Реагувати на технічні збої** — швидко локалізувати проблеми й мінімізувати час простою.

Розроблений інструмент не потребує глибоких технічних знань для базової експлуатації, що робить його доступним навіть для команд із обмеженим досвідом у галузі мережевої безпеки. Водночас гнучка архітектура дозволяє при необхідності розширити функціонал або інтегрувати додаткові модулі аналітики.

Список використаних джерел

1) Kaggle [Електронний ресурс]. – Режим доступу: https://www.kaggle.com/ – Дата звернення: 20.05.2025.
2) Основні протоколи мережі: навіщо вони використовуються [Електронний ресурс]. – Режим доступу: https://hyperhost.ua/info/uk/osnovni-protokoli-merezhi-navishcho-voni-vikoristovuyutsya – Дата звернення: 20.05.2025.
3) Типи мережевих протоколів і їх призначення (HTTP, IP, SSH, FTP, POP3, MAC) [Електронний ресурс]. – Режим доступу: https://deltahost.ua/ua/tipi-merezhevix-protokoliv-i-ih-priznachennya-http-ip-ssh-ftp-pop3-mac.html – Дата звернення: 20.05.2025.
4) Streamlit Documentation [Електронний ресурс]. – Режим доступу: https://docs.streamlit.io/ – Дата звернення: 20.05.2025.
5) Синтетичні дані: що це таке? [Електронний ресурс]. – Режим доступу: https://www.vpnunlimited.com/ua/help/cybersecurity/synthetic-data – Дата звернення: 20.05.2025.

Додаток А

```
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta
import ipaddress

# Встановлюємо seed для відтворюваності результатів
np.random.seed(42)
random.seed(82)

# Параметри аномалій
ANOMALY_CONFIG = {
    'enable_anomalies': True,     # Увімкнути/вимкнути аномалії
    'anomaly_ratio': 0.15,        # Відсоток аномальних записів у датасеті

    # Конфігурація типів аномалій
    'ddos_ratio': 0.4,            # Відсоток DDoS атак від усіх аномалій
    'misconfig_ratio': 0.2,       # Відсоток помилкових конфігурацій
    'nonstandard_port_ratio': 0.2, # Відсоток нестандартних портів
    'repeated_conn_ratio': 0.2,    # Відсоток надмірних повторюваних з'єднань

    # Параметри DDoS атаки
    'ddos_target_count': 5,       # Кількість цільових IP для DDoS
    'ddos_duration_minutes': 30,   # Тривалість однієї DDoS атаки в хвилинах
    'ddos_intensity': 'high',      # Інтенсивність: 'low', 'medium', 'high'

    # Параметри повторюваних з'єднань
    'repeat_conn_count': 50,       # Кількість повторень одного з'єднання
```

```
    'repeat_conn_interval_sec': 5, # Інтервал між повтореннями в секундах
}

# Кількість рядків у датасеті
n_rows = 23000

# Визначення категоріальних значень
protocols = ['TCP', 'UDP', 'ICMP', 'HTTP', 'HTTPS', 'DNS', 'FTP', 'SMTP', 'SSH']
services = ['http', 'https', 'dns', 'ftp', 'ssh', 'smtp', 'pop3', 'imap', 'telnet',
            'ntp', 'dhcp', 'rdp', 'vnc', '-', 'other']
states = ['FIN', 'CON', 'INT', 'RST', 'ACC', 'REQ', 'CLO', 'EST', '-']

# Стандартні порти для сервісів
standard_ports = {
    'http': 80, 'https': 443, 'dns': 53, 'ftp': 21, 'ssh': 22, 'smtp': 25,
    'pop3': 110, 'imap': 143, 'telnet': 23, 'ntp': 123, 'dhcp': 67, 'rdp': 3389, 'vnc': 5900
}

# Нестандартні порти для сервісів (для аномалій)
nonstandard_ports = {
    'http': [8080, 8888, 8008, 8081, 8000],
    'https': [8443, 9443, 4443, 8444, 9444],
    'dns': [5353, 9053, 8053],
    'ftp': [2121, 3721, 4559],
    'ssh': [2222, 2022, 922],
    'smtp': [2525, 1025, 26, 366],
    'pop3': [1110, 2110, 1109],
    'imap': [1143, 2143, 993],
    'telnet': [2323, 992],
```

```
    'ntp': [1123, 1337],
    'rdp': [3388, 13389],
    'vnc': [5901, 5902, 5800]
}

# Генерація часу початку (випадково за останній місяць)
base_time = datetime.now() - timedelta(days=30)
start_times = [base_time + timedelta(seconds=random.randint(0, 30*24*60*60)) for
_ in range(n_rows)]

# Створення базового DataFrame
df = pd.DataFrame({
    'id': range(1, n_rows + 1),
    'proto': np.random.choice(protocols, size=n_rows, p=[0.4, 0.3, 0.1, 0.05, 0.05,
0.04, 0.02, 0.02, 0.02]),
    'service': np.random.choice(services, size=n_rows, p=[0.25, 0.2, 0.15, 0.1, 0.1,
0.05, 0.05, 0.03, 0.02, 0.01, 0.01, 0.01, 0.01, 0.005, 0.005]),
    'state': np.random.choice(states, size=n_rows),
    'dur': np.random.exponential(10, n_rows),  # Тривалість у секундах
(експоненційний розподіл)
    'spkts': np.random.randint(1, 1000, n_rows),  # Відправлені пакети
    'dpkts': np.random.randint(1, 1000, n_rows),  # Отримані пакети
    'sbytes': np.random.randint(100, 10000000, n_rows),  # Відправлені байти
    'dbytes': np.random.randint(100, 10000000, n_rows),  # Отримані байти
    'rate': np.random.exponential(1000, n_rows),  # Швидкість передачі даних
(біти/с)
    'sttl': np.random.randint(30, 255, n_rows),  # Source TTL
    'dttl': np.random.randint(30, 255, n_rows),  # Destination TTL
    'sload': np.random.exponential(5, n_rows),  # Source load
    'dload': np.random.exponential(5, n_rows),  # Destination load
```

```
    'sloss': np.random.randint(0, 100, n_rows),  # Source loss (packets)
    'dloss': np.random.randint(0, 100, n_rows),  # Destination loss (packets)
    'sinpkt': np.random.exponential(0.01, n_rows),  # Source inter-packet arrival time
    'dinpkt': np.random.exponential(0.01, n_rows),  # Destination inter-packet arrival time
    'sjit': np.random.exponential(0.005, n_rows),  # Source jitter
    'djit': np.random.exponential(0.005, n_rows),  # Destination jitter
    'swin': np.random.randint(1000, 65535, n_rows),  # Source window size
    'dwin': np.random.randint(1000, 65535, n_rows),  # Destination window size
    'stcpb': np.random.randint(100000, 1000000000, n_rows),  # Source TCP base sequence number
    'dtcpb': np.random.randint(100000, 1000000000, n_rows),  # Destination TCP base sequence number
    'tcprtt': np.random.exponential(0.1, n_rows),  # TCP connection round trip time
    'synack': np.random.exponential(0.05, n_rows),  # TCP connection setup time
    'ackdat': np.random.exponential(0.05, n_rows),  # TCP connection setup time
    'is_attack': np.random.choice([0, 1], size=n_rows, p=[0.9, 0.1]),  # Флаг атаки (0-нормальний трафік, 1-атака)
})

# Генерація IP-адрес
def generate_random_ip():
    return f"{random.randint(1, 223)}.{random.randint(0, 255)}.{random.randint(0, 255)}.{random.randint(1, 254)}"

# Розширений список країн для IP-геолокації
countries = [
    'Україна', 'США', 'Німеччина', 'Польща', 'Франція', 'Китай', 'Велика Британія', 'Канада',
    'Індія', 'Японія', 'Австралія', 'Бразилія', 'Південна Корея', 'Італія', 'Іспанія',
    'Нідерланди', 'Швеція', 'Сінгапур', 'Ізраїль', 'ОАЕ'
```

```
]

# Різні ваги для розподілу трафіку за країнами (додаємо високий трафік з Індії)
country_weights = [
    0.25, 0.12, 0.08, 0.06, 0.05, 0.05, 0.05, 0.04,  # Original countries with adjusted weights
    0.10, 0.03, 0.02, 0.03, 0.02, 0.02, 0.02,  # Added countries with various weights (Індія має високий трафік)
    0.02, 0.01, 0.01, 0.01, 0.01
]

# Make sure weights sum to 1.0
country_weights = [w/sum(country_weights) for w in country_weights]

# Додамо IP-адреси і країни до датафрейму
df['src_ip'] = [generate_random_ip() for _ in range(n_rows)]
df['dst_ip'] = [generate_random_ip() for _ in range(n_rows)]
df['src_country'] = np.random.choice(countries, size=n_rows, p=country_weights)
df['dst_country'] = np.random.choice(countries, size=n_rows, p=country_weights)

# Додаємо порти (стандартні для початку)
df['src_port'] = np.random.randint(1024, 65535, n_rows)  # Динамічні порти джерела
df['dst_port'] = df['service'].apply(lambda s: standard_ports.get(s, random.randint(1, 1023)))

# Define different weights for source and destination countries
src_country_weights = [
    0.3, 0.1, 0.05, 0.08, 0.05, 0.05, 0.03, 0.02,  # First 8 countries
    0.04, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03,      # Next 7 countries
```

```
    0.02, 0.02, 0.02, 0.02, 0.02                # Last 5 countries
]

dst_country_weights = [
    0.15, 0.15, 0.1, 0.08, 0.08, 0.05, 0.04, 0.05, # First 8 countries
    0.04, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03,      # Next 7 countries
    0.02, 0.02, 0.02, 0.01, 0.01                # Last 5 countries
]

# Ensure weights sum to 1.0
src_country_weights = [w/sum(src_country_weights) for w in src_country_weights]
dst_country_weights = [w/sum(dst_country_weights) for w in dst_country_weights]

# Apply different distributions
df['src_country'] = np.random.choice(countries, size=n_rows,
p=src_country_weights)
df['dst_country'] = np.random.choice(countries, size=n_rows,
p=dst_country_weights)

# Create correlation between countries (e.g., Ukraine often communicates with
Poland)
mask_ukraine = df['src_country'] == 'Україна'
poland_index = np.random.choice(np.where(mask_ukraine)[0],
size=int(sum(mask_ukraine)*0.4))
df.loc[poland_index, 'dst_country'] = 'Польща'

# Add byte volume differences (more bytes sent from certain countries)
multiplier = df['src_country'].map({
    'Україна': 1.0, 'США': 1.5, 'Німеччина': 1.2, 'Польща': 0.8,
    'Франція': 0.9, 'Китай': 1.7, 'Велика Британія': 1.3, 'Канада': 0.7
```

```
})
df['sbytes'] = df['sbytes'] * multiplier

# Додавання часових міток
df['start_time'] = start_times
df['end_time'] = [st + timedelta(seconds=dur) for st, dur in zip(start_times, df['dur'])]

# Замість генерації міток аномалій, створюємо записи з нетиповими значеннями
def generate_anomalies_without_labeling(df):
    # Кількість аномальних записів
    anomaly_count = int(len(df) * ANOMALY_CONFIG['anomaly_ratio'])

    # Вибираємо випадкові індекси для модифікації
    anomaly_indices = np.random.choice(df.index, size=anomaly_count,
replace=False)

    # Розділяємо аномальні записи на категорії
    ddos_count = int(anomaly_count * ANOMALY_CONFIG['ddos_ratio'])
    misconfig_count = int(anomaly_count * ANOMALY_CONFIG['misconfig_ratio'])
    nonstandard_port_count = int(anomaly_count *
ANOMALY_CONFIG['nonstandard_port_ratio'])

    # Індекси для різних типів аномалій
    ddos_indices = anomaly_indices[:ddos_count]
    misconfig_indices = anomaly_indices[ddos_count:ddos_count+misconfig_count]
    nonstandard_port_indices =
anomaly_indices[ddos_count+misconfig_count:ddos_count+misconfig_count+nonsta
ndard_port_count]

    # 1. DDoS-подібні записи - завищені показники пакетів та швидкості
```

```
    for idx in ddos_indices:
        # Висока швидкість пакетів
        df.at[idx, 'spkts'] = random.randint(3000, 10000)
        # Короткі з'єднання
        df.at[idx, 'dur'] = random.uniform(0.0001, 0.01)
        # Малі розміри пакетів
        df.at[idx, 'sbytes'] = df.at[idx, 'spkts'] * random.randint(1, 5)
        # Висока швидкість
        df.at[idx, 'rate'] = df.at[idx, 'spkts'] / df.at[idx, 'dur'] if df.at[idx, 'dur'] > 0 else
9999999

    # 2. Неправильні конфігурації
    for idx in misconfig_indices:
        misconfig_type = random.choice(['wrong_ttl', 'window_size', 'packet_size'])

        if misconfig_type == 'wrong_ttl':
            # Нетипові TTL значення
            df.at[idx, 'sttl'] = random.choice([1, 2, 255, 254])
            df.at[idx, 'dttl'] = random.choice([1, 2, 255, 254])

        elif misconfig_type == 'window_size':
            # Нетипові розміри вікна TCP
            df.at[idx, 'swin'] = random.choice([1, 2, 3, 65535, 65534])
            df.at[idx, 'dwin'] = random.choice([1, 2, 3, 65535, 65534])

        elif misconfig_type == 'packet_size':
            # Невідповідність пакетів і байтів
            df.at[idx, 'spkts'] = random.randint(500, 1000)
            df.at[idx, 'sbytes'] = random.randint(500, 1000)  # ~1 байт на пакет
```

```
    # 3. Нестандартні порти
    for idx in nonstandard_port_indices:
        service = df.at[idx, 'service']
        if service in nonstandard_ports and nonstandard_ports[service]:
            df.at[idx, 'dst_port'] = random.choice(nonstandard_ports[service])
        else:
            df.at[idx, 'dst_port'] = random.randint(10000, 65535)

    return df

# Використовуємо нову функцію замість старої
# df = generate_anomalies(df)  # Коментуємо стару функцію
df = generate_anomalies_without_labeling(df)  # Використовуємо нову функцію

# Видаляємо колонки anomaly та anomaly_type
if 'anomaly' in df.columns:
    df = df.drop('anomaly', axis=1)
if 'anomaly_type' in df.columns:
    df = df.drop('anomaly_type', axis=1)

# Генеруємо аномалії
df = generate_anomalies_without_labeling(df)

# Перетворення даних та обмеження для підвищення реалізму
# Округлення значень з плаваючою крапкою до 6 знаків після коми
for col in ['dur', 'rate', 'sload', 'dload', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'tcprtt', 'synack',
'ackdat']:
    df[col] = df[col].round(6)
```

```
# Збереження даних в CSV
output_path = "data/dataset1.csv"
df.to_csv(output_path, index=False)

print(f"Створено датасет з {len(df)} рядками у файлі {output_path}")
print(f"Нормальні з'єднання: {(df['anomaly'] == 0).sum()}")
print(f"Аномальні з'єднання: {(df['anomaly'] == 1).sum()}")
print(f"Типи аномалій: {df[df['anomaly'] ==
1]['anomaly_type'].value_counts().to_dict()}")
print(f"Колонки: {', '.join(df.columns)}")
```

Додаток Б

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from src.data_loader import load_dataset
from src.data_cleaner import clean_data
import ipaddress

# Визначаємо колонки з числовими даними для аналізу
NUMERIC_COLUMNS = ['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate',
                   'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss',
                   'sinpkt', 'dinpkt', 'sjit', 'djit', 'tcprtt', 'synack', 'ackdat']

def main():
    st.set_page_config(layout="wide", page_title="Аналіз мережевого трафіку")

    st.title("Інтерактивна панель аналізу мережевого трафіку")

    # Load and process data
    with st.spinner("Завантаження даних..."):
```

```
    df_real = load_dataset('real')
    df_synthetic = load_dataset()
    df = df_synthetic
    df = clean_data(df, NUMERIC_COLUMNS)

dataset_option = st.sidebar.selectbox(
    "Оберіть набір даних:",
    options=["Синтетичні дані", "Реальні дані"]
)

if dataset_option == "Реальні дані":
    df = df_real
else:
    df = df_synthetic

# Додамо обробку timestamp колонок
if 'start_time' in df.columns:
    df['start_time'] = pd.to_datetime(df['start_time'])
    df['hour'] = df['start_time'].dt.hour
    df['date'] = df['start_time'].dt.date

# Sidebar for filters
st.sidebar.header("Фільтри")
```

```
protocols = ['Всі'] + sorted(df['proto'].unique().tolist())
selected_protocol = st.sidebar.selectbox("Протокол:", protocols)

services = ['Всі'] + sorted(df['service'].unique().tolist())
selected_service = st.sidebar.selectbox("Сервіс:", services)

states = ['Всі'] + sorted(df['state'].unique().tolist())
selected_state = st.sidebar.selectbox("Стан:", states)

# Filter by date range
if 'date' in df.columns:
    date_min = df['date'].min()
    date_max = df['date'].max()
    selected_date_range = st.sidebar.date_input(
        "Діапазон дат:",
        value=(date_min, date_max),
        min_value=date_min,
        max_value=date_max
    )

    # Ensure we have both start and end dates
    if len(selected_date_range) == 2:
```

```
        start_date, end_date = selected_date_range
        date_mask = (df['date'] >= start_date) & (df['date'] <= end_date)
    else:
        date_mask = df['date'] == selected_date_range[0]
else:
    date_mask = pd.Series(True, index=df.index)

# New filter specifically for anomalies
st.sidebar.header("Фільтр аномалій")

if 'anomaly' in df.columns:
    anomaly_options = ['Всі дані', 'Тільки нормальні', 'Тільки аномалії']
    anomaly_filter = st.sidebar.radio("Показати:", anomaly_options)
else:
    anomaly_filter = 'Всі дані'

# Apply filters to dataframe
filtered_df = df.copy()

if selected_protocol != 'Всі':
    filtered_df = filtered_df[filtered_df['proto'] == selected_protocol]

if selected_service != 'Всі':
```

```
    filtered_df = filtered_df[filtered_df['service'] == selected_service]

if selected_state != 'Всі':
    filtered_df = filtered_df[filtered_df['state'] == selected_state]

# Apply date filter
filtered_df = filtered_df[date_mask]

# Apply anomaly filter
if 'anomaly' in filtered_df.columns and anomaly_filter != 'Всі дані':
    if anomaly_filter == 'Тільки нормальні':
        filtered_df = filtered_df[filtered_df['anomaly'] == 0]
    else:  # 'Тільки аномалії'
        filtered_df = filtered_df[filtered_df['anomaly'] == 1]

# Display basic statistics
st.subheader("Основна статистика")
col1, col2, col3, col4 = st.columns(4)

col1.metric("Кількість сесій", f"{len(filtered_df):,}")
col2.metric("Середня тривалість", f"{filtered_df['dur'].mean():.2f} с")
col3.metric("Загальний обсяг даних", f"{filtered_df['sbytes'].sum():,} байтів")
```

```
    col4.metric("Найпоширеніший протокол", filtered_df['proto'].value_counts().index[0])

    # Create tabs for different visualizations
    tab1, tab2, tab3, tab4, tab5, tab6, tab7 = st.tabs(["Розподіли", "Кореляції", "Протоколи",
                                                        "Загальний огляд", "Часовий аналіз",
                                                        "Геовізуалізація", "Виявлення аномалій"])

    with tab1:
        st.subheader("Розподіл числових показників")
        column = st.selectbox("Оберіть показник:", options=NUMERIC_COLUMNS)
        fig = px.histogram(filtered_df, x=column, nbins=50, marginal="box",
                           color='anomaly' if 'anomaly' in filtered_df.columns else None,
                           color_discrete_map={0: 'blue', 1: 'red'})
        st.plotly_chart(fig, use_container_width=True)

    with tab2:
        st.subheader("Кореляційна матриця")
        corr = filtered_df[NUMERIC_COLUMNS].corr()
        fig = px.imshow(corr, text_auto=True, color_continuous_scale="RdBu_r")
        st.plotly_chart(fig, use_container_width=True)
```

```
with tab3:
    st.subheader("Аналіз протоколів")
    col1, col2 = st.columns(2)

    with col1:
        proto_counts = filtered_df['proto'].value_counts()
        fig = px.pie(values=proto_counts.values, names=proto_counts.index,
                     title="Розподіл протоколів")
        st.plotly_chart(fig, use_container_width=True)

    with col2:
        service_counts = filtered_df['service'].value_counts().head(10)
        fig = px.bar(x=service_counts.index, y=service_counts.values,
                     title="Топ-10 сервісів", labels={'x': 'Сервіс', 'y': 'Кількість'})
        st.plotly_chart(fig, use_container_width=True)

with tab4:
    st.subheader("Загальний огляд трафіку")

    # Scatter plot of bytes vs packets
    fig = px.scatter(filtered_df, x="spkts", y="sbytes",
                     size="dur", color="proto", hover_name="service",
                     log_x=True, log_y=True,
```

```
                labels={"spkts": "Пакети", "sbytes": "Байти", "dur": "Тривалість"},
                title="Співвідношення пакетів та байт за протоколами")
    st.plotly_chart(fig, use_container_width=True)

    # Boxplot of duration by protocol
    fig = px.box(filtered_df, x="proto", y="dur",
                 color="proto",
                 labels={"proto": "Протокол", "dur": "Тривалість (с)"},
                 title="Розподіл тривалості з'єднань за протоколами")
    st.plotly_chart(fig, use_container_width=True)

with tab5:
    st.subheader("Часовий аналіз")

    if 'hour' in filtered_df.columns:
        # Hourly traffic volume
        hourly_traffic = filtered_df.groupby('hour')['sbytes'].sum().reset_index()
        fig = px.line(hourly_traffic, x='hour', y='sbytes',
                      labels={'hour': 'Година доби', 'sbytes': 'Обсяг даних'},
                      title="Розподіл трафіку за годинами доби")
        st.plotly_chart(fig, use_container_width=True)

        # Теплова карта навантаження за днями тижня і годинами
```

```
        df['day_of_week'] = pd.to_datetime(df['start_time']).dt.dayofweek
        day_hour_traffic = df.groupby(['day_of_week', 'hour'])['sbytes'].sum().reset_index()
        day_hour_pivot = day_hour_traffic.pivot(index='day_of_week', columns='hour', values='sbytes')

        days = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', "П'ятниця", 'Субота', 'Неділя']
        fig2 = px.imshow(day_hour_pivot,
                         labels=dict(x="Година доби", y="День тижня", color="Обсяг даних"),
                         y=days,
                         color_continuous_scale="Viridis")
        st.plotly_chart(fig2, use_container_width=True)

        # Визначення піків навантаження
        peak_hours = hourly_traffic.nlargest(3, 'sbytes')
        st.subheader("Пікові години навантаження")
        st.write(f"Години з найбільшим трафіком: {', '.join(map(str, peak_hours['hour'].tolist()))}")

    with tab6:
        st.subheader("Географічна візуалізація трафіку")
```

```
# Вибір напряму трафіку для аналізу
traffic_direction = st.radio(
    "Напрямок трафіку:",
    options=["Джерело", "Призначення"],
    horizontal=True
)

# Словник для перетворення українських назв країн в ISO коди
country_iso_map = {
    'Україна': 'UKR',
    'США': 'USA',
    'Німеччина': 'DEU',
    'Польща': 'POL',
    'Франція': 'FRA',
    'Китай': 'CHN',
    'Велика Британія': 'GBR',
    'Канада': 'CAN',
    # Додані країни
    'Індія': 'IND',
    'Японія': 'JPN',
    'Австралія': 'AUS',
    'Бразилія': 'BRA',
    'Південна Корея': 'KOR',
```

```
        'Італія': 'ITA',
        'Іспанія': 'ESP',
        'Нідерланди': 'NLD',
        'Швеція': 'SWE',
        'Сінгапур': 'SGP',
        'Ізраїль': 'ISR',
        'ОАЕ': 'ARE'
    }

    country_col = 'src_country' if traffic_direction == "Джерело" else 'dst_country'

    # Агрегація даних за країнами
    country_traffic = filtered_df.groupby(country_col).agg(
        total_bytes=pd.NamedAgg(column='sbytes', aggfunc='sum'),
        count=pd.NamedAgg(column='id', aggfunc='count')
    ).reset_index()

    # Додаємо ISO коди для хороплету
    country_traffic['iso_alpha'] = country_traffic[country_col].map(country_iso_map)

    # Створюємо хороплет
    fig = px.choropleth(country_traffic,
```

```
                locations="iso_alpha",
                color="total_bytes",
                hover_name=country_col,
                title=f"Розподіл трафіку за країнами ({traffic_direction.lower()})",
                color_continuous_scale=px.colors.sequential.Plasma)

st.plotly_chart(fig, use_container_width=True)

# Таблиця з детальною інформацією
st.subheader("Деталі трафіку за країнами")
country_traffic['total_mb'] = country_traffic['total_bytes'] / (1024 * 1024)
country_traffic = country_traffic.sort_values('total_bytes', ascending=False)
st.dataframe(
    country_traffic.rename(columns={
        country_col: 'Країна',
        'count': 'Кількість з\'єднань',
        'total_bytes': 'Загальний обсяг (байти)',
        'total_mb': 'Загальний обсяг (МБ)'
    }).round(2)
)

# Додаємо аналіз характеристик трафіку за країнами
st.subheader("Характеристики трафіку за країнами")
```

```
metrics_options = ['sbytes', 'dbytes', 'spkts', 'dpkts', 'dur', 'sloss', 'dloss', 'sjit', 'djit']

selected_metric = st.selectbox("Оберіть метрику для порівняння:", options=metrics_options)


# Розрахунок середнього значення метрики за країнами

country_metrics = filtered_df.groupby(country_col)[selected_metric].mean().reset_index()

country_metrics = country_metrics.sort_values(selected_metric, ascending=False)


# Горизонтальна діаграма для порівняння метрики за країнами

fig = px.bar(country_metrics,

        x=selected_metric,

        y=country_col,

        orientation='h',

        title=f"Середнє значення {selected_metric} за країнами",

        labels={country_col: "Країна", selected_metric: f"Середнє значення {selected_metric}"})


st.plotly_chart(fig, use_container_width=True)


# Додаємо інтерактивну довідку про особливості трафіку різних країн

st.info("""
```

```
    ### Особливості мережевого трафіку за країнами
    - **Індія**: Вищий обсяг даних (на 80% більше байтів)
    - **Китай**: Більша кількість пакетів
    - **США**: Триваліші з'єднання
    - **Бразилія**: Вища втрата пакетів
    - **Європейські країни**: Нижчий джитер (затримка)
    """)

with tab7:
    st.subheader("Виявлення та аналіз аномалій")

    if 'anomaly' in df.columns:
        # Display overview of anomalies
        anomaly_counts = df['anomaly'].value_counts()
        total_connections = len(df)
        anomaly_percent = (anomaly_counts.get(1, 0) / total_connections) * 100 if total_connections > 0 else 0

        st.write(f"### Загальна статистика аномалій")
        st.write(f"Всього з'єднань: {total_connections}")
        st.write(f"Нормальні з'єднання: {anomaly_counts.get(0, 0)} ({100 - anomaly_percent:.2f}%)")
        st.write(f"Аномальні з'єднання: {anomaly_counts.get(1, 0)} ({anomaly_percent:.2f}%)")
```

```
        # Distribution of anomaly types
        if 'anomaly_type' in df.columns and anomaly_counts.get(1, 0) > 0:
            anomaly_types = df[df['anomaly'] == 1]['anomaly_type'].value_counts()

            fig = px.pie(
                values=anomaly_types.values,
                names=anomaly_types.index,
                title="Розподіл типів аномалій"
            )
            st.plotly_chart(fig, use_container_width=True)

            # Create detailed analysis by anomaly type
            st.write("### Детальний аналіз аномалій за типами")

            # Get unique anomaly types
            anomaly_type_options = ['Всі типи'] + sorted(df[df['anomaly'] == 1]['anomaly_type'].unique().tolist())
            selected_anomaly_type = st.selectbox("Оберіть тип аномалії для аналізу:", anomaly_type_options)

            # Filter data based on selected anomaly type
            if selected_anomaly_type != 'Всі типи':
```

```
            anomaly_data = df[(df['anomaly'] == 1) & (df['anomaly_type'] == selected_anomaly_type)]
        else:
            anomaly_data = df[df['anomaly'] == 1]

        # Calculate some statistics for the selected anomaly type
        if not anomaly_data.empty:
            col1, col2, col3 = st.columns(3)

            col1.metric("Кількість з'єднань", len(anomaly_data))
            col2.metric("Середня тривалість", f"{anomaly_data['dur'].mean():.4f} с")
            col3.metric("Середній розмір", f"{int(anomaly_data['sbytes'].mean())} байт")

            # Compare normal vs anomaly characteristics
            st.write("### Порівняння нормального та аномального трафіку")

            comparison_metric = st.selectbox(
                "Оберіть метрику для порівняння:",
                options=['pkt_to_byte_ratio', 'connection_rate', 'duration'] + NUMERIC_COLUMNS
            )
```

```
            # Calculate metrics for comparison
            if comparison_metric == 'pkt_to_byte_ratio':
                normal_data = df[df['anomaly'] == 0].copy()
                normal_data['pkt_to_byte_ratio'] = normal_data['spkts'] / (normal_data['sbytes'] + 1)
                anomaly_data['pkt_to_byte_ratio'] = anomaly_data['spkts'] / (anomaly_data['sbytes'] + 1)

                fig = go.Figure()

fig.add_trace(go.Histogram(x=normal_data['pkt_to_byte_ratio'].clip(upper=0.1),
                                   name='Нормальний трафік', opacity=0.7,
                                   marker_color='blue'))

fig.add_trace(go.Histogram(x=anomaly_data['pkt_to_byte_ratio'].clip(upper=0.1),
                                   name='Аномальний трафік', opacity=0.7,
                                   marker_color='red'))

                fig.update_layout(title='Порівняння відношення пакетів до байтів',
                                  xaxis_title='Пакетів на байт (обмежено до 0.1)',
                                  yaxis_title='Кількість з\'єднань',
                                  barmode='overlay')
                st.plotly_chart(fig, use_container_width=True)
```

```
            elif comparison_metric == 'connection_rate':
                if 'start_time' in df.columns:
                    # Group connections by 5-minute intervals
                    df['time_window'] = df['start_time'].dt.floor('5min')
                    normal_rate = df[df['anomaly'] == 0].groupby('time_window').size()
                    anomaly_rate = df[df['anomaly'] == 1].groupby('time_window').size()

                    # Combine into one dataframe for visualization
                    rate_df = pd.DataFrame({
                        'time': normal_rate.index,
                        'Нормальний трафік': normal_rate.values
                    })

                    # Add anomaly rate if it exists for that time window
                    for time, count in anomaly_rate.items():
                        if time in rate_df['time'].values:
                            rate_df.loc[rate_df['time'] == time, 'Аномальний трафік'] = count
                        else:
                            new_row = pd.DataFrame({'time': [time], 'Нормальний трафік': [0], 'Аномальний трафік': [count]})
                            rate_df = pd.concat([rate_df, new_row], ignore_index=True)
```

```
                rate_df = rate_df.fillna(0).sort_values('time')

                fig = px.line(rate_df, x='time', y=['Нормальний трафік', 'Аномальний трафік'],
                              title='Інтенсивність з\'єднань з часом',
                              labels={'value': 'Кількість з\'єднань', 'time': 'Час',
                                      'variable': 'Тип трафіку'})
                st.plotly_chart(fig, use_container_width=True)

            elif comparison_metric == 'duration':
                fig = go.Figure()
                fig.add_trace(go.Box(y=df[df['anomaly'] == 0]['dur'].clip(upper=20),
                                     name='Нормальний трафік',
                                     marker_color='blue'))
                fig.add_trace(go.Box(y=anomaly_data['dur'].clip(upper=20),
                                     name='Аномальний трафік',
                                     marker_color='red'))
                fig.update_layout(title='Порівняння тривалості з\'єднань',
                                  yaxis_title='Тривалість (с, обмежено до 20с)')
                st.plotly_chart(fig, use_container_width=True)

            else:  # For standard numeric metrics
                fig = go.Figure()
```

```
                fig.add_trace(go.Box(y=df[df['anomaly'] == 0][comparison_metric],
                                     name='Нормальний трафік',
                                     marker_color='blue'))
                fig.add_trace(go.Box(y=anomaly_data[comparison_metric],
                                     name='Аномальний трафік',
                                     marker_color='red'))
                fig.update_layout(title=f'Порівняння {comparison_metric}',
                                  yaxis_title=f'{comparison_metric}')
                st.plotly_chart(fig, use_container_width=True)

            # Show examples of anomalies
            st.write("### Приклади аномальних з'єднань")
            sample_size = min(10, len(anomaly_data))
            display_cols = ['start_time', 'proto', 'service', 'dur', 'spkts', 'sbytes', 'src_ip', 'dst_ip', 'anomaly_type']
            st.dataframe(anomaly_data[display_cols].sample(sample_size))

        else:
            # Run anomaly detection if not already present
            st.write("### Виявлення аномалій в датасеті")
            st.write("У цьому датасеті відсутні мітки аномалій. Натисніть кнопку нижче, щоб запустити алгоритми виявлення аномалій:")
```

```
if st.button("Виявити аномалії"):
    with st.spinner("Аналіз даних і виявлення аномалій..."):
        # Initialize anomaly column if it doesn't exist
        if 'anomaly' not in df.columns:
            df['anomaly'] = 0
        if 'anomaly_type' not in df.columns:
            df['anomaly_type'] = ''

        # 1. Detect DDoS attacks (existing code)
        st.write("1. Виявлення атак DDoS...")

        # Group by destination IP and time window
        df['time_window'] = pd.to_datetime(df['start_time']).dt.floor('10min')
        ddos_threshold_pkt_rate = 1000  # Поріг для швидкості пакетів
        # Шукаємо з'єднання з надзвичайно високою швидкістю пакетів
        ddos_mask = (df['spkts'] / df['dur'] > ddos_threshold_pkt_rate) & (df['dur'] < 0.1)
        df.loc[ddos_mask, 'anomaly'] = 1
        df.loc[ddos_mask, 'anomaly_type'] = 'ddos'

        # 2. Detect misconfigurations (enhanced)
        st.write("2. Виявлення помилкових конфігурацій...")
```

```
# a. Неправильні TTL значення
wrong_ttl_mask = ((df['sttl'] <= 2) | (df['sttl'] >= 254) |
                  (df['dttl'] <= 2) | (df['dttl'] >= 254))
df.loc[wrong_ttl_mask, 'anomaly'] = 1
df.loc[wrong_ttl_mask, 'anomaly_type'] = 'misconfig:wrong_ttl'

# b. Нетипові розміри вікна TCP
window_size_mask = ((df['swin'] <= 3) | (df['swin'] >= 65534) |
                    (df['dwin'] <= 3) | (df['dwin'] >= 65534))
df.loc[window_size_mask, 'anomaly'] = 1
df.loc[window_size_mask, 'anomaly_type'] = 'misconfig:window_size'

# c. Аномальне співвідношення пакетів до байтів
pkt_byte_ratio_mask = ((df['spkts'] > 100) & (df['sbytes'] / df['spkts'] <
10))
df.loc[pkt_byte_ratio_mask, 'anomaly'] = 1
df.loc[pkt_byte_ratio_mask, 'anomaly_type'] = 'misconfig:packet_size'

# 3. Detect nonstandard ports
st.write("3. Виявлення нестандартних портів...")

# Створюємо словник відповідності сервісів та стандартних портів
service_port_map = {
```

```
                'http': 80, 'https': 443, 'dns': 53, 'ftp': 21, 'ssh': 22, 'smtp': 25,
                'pop3': 110, 'imap': 143, 'telnet': 23, 'ntp': 123, 'rdp': 3389
            }

            # Перевіряємо порти на відповідність сервісу
            for service, port in service_port_map.items():
                nonstandard_port_mask = (df['service'] == service) & (df['dst_port'] != port)
                df.loc[nonstandard_port_mask, 'anomaly'] = 1
                df.loc[nonstandard_port_mask, 'anomaly_type'] = 'nonstandard_port'

            st.success(f"Аналіз завершено! Виявлено {df['anomaly'].sum()} аномалій.")
            st.write(f"Розподіл аномалій за типами:")

            anomaly_counts = df[df['anomaly'] == 1]['anomaly_type'].value_counts()
            st.write(anomaly_counts)
        else:
            st.write("Колонка 'anomaly' відсутня у датасеті. Для виявлення аномалій потрібно оновити структуру даних.")

if __name__ == "__main__":
    main()
```

Додаток В

data_cleaner.py

```
import pandas as pd
def clean_data(df, numeric_columns):
    df = df.dropna()
    for col in numeric_columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
    df = df.dropna()  # Після приведення типів ще раз чистимо
    return df
```

data_loader.py

```
import pandas as pd
import os
def load_dataset(dataset_type='synthetic'):
    base_path = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    data_path = os.path.join(base_path, "data")

    if dataset_type.lower() == 'real':
        file_path = os.path.join(data_path, "dataset.csv")
        print(f"Loading real dataset from {file_path}")
    else:
```

```
        file_path = os.path.join(data_path, "dataset1.csv")
        print(f"Loading synthetic dataset from {file_path}")

    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Dataset file not found: {file_path}")

    return pd.read_csv(file_path)

def load_data(file_path=None):

    if file_path is None:
        # Default to synthetic data if no path specified
        return load_dataset('synthetic')
    else:
        if not os.path.exists(file_path):
            raise FileNotFoundError(f"Data file not found: {file_path}")
        return pd.read_csv(file_path)
```