МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ВИШНЕВСЬКИЙ АРТУР ВАЛЕРІЙОВИЧ

Допускається до захисту:

в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____ О. В. Зелінська

«_____» _____ 2024 р.

ІНФОРМАЦІЙНА СИСТЕМА ПРОГНОЗУВАННЯ ЗАБРУДНЕННЯ ПОВІТРЯ ЗА ДОПОМОГОЮ АНАЛІЗУ МЕТЕОРОЛОГІЧНИХ ТА ЕКОЛОГІЧНИХ ДАНИХ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:

Н. А. Потапова, доцент кафедри
інформаційних технологій,
канд. екон. наук, доцент
_____
(Підпис)

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

Голова ЕК: _____
(Підпис)

Вінниця – 2024

АНОТАЦІЯ

Вишневський А.В. Розробка інформаційної системи прогнозування забруднення повітря за допомогою аналізу метеорологічних та екологічних даних. Спеціальність 122 «Комп'ютерні науки». Освітня програма Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Магістерська робота присвячена розробці інформаційної системи для моніторингу, аналізу та прогнозування показників забруднення повітря. У дослідженні використано сучасні підходи до обробки великих даних, методи машинного навчання та хмарні технології для реалізації системи. Особливу увагу приділено інтеграції екологічних та метеорологічних даних, розробці моделей прогнозування та їх розгортанню в реальному часі. Основними інструментами розробки стали сервіси Amazon Web Services (AWS), бібліотеки Python (pandas, scikit-learn, Matplotlib) та середовище Jupyter Labs.

Робота складається зі вступу, трьох розділів, висновків та додатків. У першому розділі розглянуто теоретичні аспекти прогнозування забруднення повітря, зокрема статистичні, емпіричні та моделі на основі штучного інтелекту. Другий розділ присвячено розробці інформаційної інфраструктури, що включає автоматизований збір даних з API, їх обробку за допомогою AWS Lambda та зберігання у структурованому вигляді в S3-бакетах. У третьому розділі проведено аналіз часових рядів, кореляційний аналіз між параметрами забруднення та навчання моделі Random Forest для прогнозування індексу якості повітря (AQI). Розгорнута модель інтегрована у середовище AWS SageMaker з можливістю доступу до прогнозів через RESTful API.

Магістерська робота складається зі вступу, 3 розділів, висновків, списку літератури з 40 джерел та 4 рисунків. Загальний обсяг роботи становить 61 сторінку.

ABSTRACT

Vyshnevsky A.V. Development of an information system for air pollution forecasting using meteorological and environmental data analysis. Specialty 122 "Computer Science". Educational program Computer Data Processing (Data Science). Vasyl Stus Donetsk National University, Vinnytsia 2024.

The master's thesis is devoted to the development of an information system for monitoring, analyzing and forecasting air pollution. The study used modern approaches to big data processing, machine learning methods and cloud technologies to implement the system. Particular attention is paid to the integration of environmental and meteorological data, the development of forecasting models and their deployment in real time. The main development tools were Amazon Web Services (AWS), Python libraries (pandas, scikit-learn, Matplotlib) and the Jupyter Labs environment.

The work consists of an introduction, three sections, conclusions and appendices. The first section considers theoretical aspects of air pollution forecasting, in particular statistical, empirical and models based on artificial intelligence. The second section is devoted to the development of an information infrastructure, which includes automated data collection from APIs, their processing using AWS Lambda and storage in a structured form in S3 buckets. The third section provides time series analysis, correlation analysis between pollution parameters and training of a Random Forest model for predicting the air quality index (AQI). The deployed model is integrated into the AWS SageMaker environment with the ability to access forecasts via a RESTful API.

The master's thesis consists of an introduction, 3 chapters, conclusions, a list of literature from 40 sources and 4 figures. The total volume of the work is 61 pages.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AQI – індекс якості повітря (Air Quality Index)

AWS – хмарні сервіси Amazon Web Services

CSV – формат текстового файлу для зберігання табличних даних (Comma-Separated Values)

EDA – дослідницький аналіз даних (Exploratory Data Analysis)

API – інтерфейс прикладного програмування (Application Programming Interface)

S3 – сервіс зберігання об’єктів у хмарі Amazon (Simple Storage Service)

IAM – управління ідентифікацією та доступом (Identity and Access Management)

PM2.5 – дрібнодисперсні частки діаметром менше 2,5 мікрон

NO_2 – діоксид азоту

SO_2 – діоксид сірки

CO – чадний газ

Jupyter Labs – інтерактивне середовище для аналізу даних і програмування

Python – мова програмування

MAE – середня абсолютна похибка (Mean Absolute Error)

RMSE – корінь середнього квадрату похибок (Root Mean Squared Error)

ML – машинне навчання (Machine Learning)

Random Forest – алгоритм ансамблевого машинного навчання

SageMaker – сервіс AWS для навчання, розгортання та керування моделями машинного навчання

Lambda – безсерверний обчислювальний сервіс AWS

Folium – бібліотека Python для створення інтерактивних карт

Scikit-learn – бібліотека Python для машинного навчання

Matplotlib – бібліотека Python для побудови графіків

Seaborn – бібліотека Python для візуалізації даних

ВСТУП

Актуальність теми. У сучасному світі, де екологічні проблеми стають дедалі гострішими, важливу роль відіграють інформаційні технології для моніторингу та аналізу забруднення навколишнього середовища. Забруднення повітря є однією з найважливіших екологічних проблем, оскільки воно безпосередньо впливає на здоров'я людей, екосистеми та клімат. Ефективне прогнозування рівнів забруднення повітря є ключовим фактором для ухвалення рішень у сфері екологічного управління та забезпечення безпеки населення.

Основна проблема, з якою стикається сучасне суспільство, полягає у відсутності інтегрованих та автоматизованих систем, які могли б забезпечити точний аналіз і прогнозування забруднення повітря в реальному часі. Використання таких систем дозволяє оптимізувати процеси моніторингу, забезпечити своєчасну реакцію на загрози та знизити ризики для здоров'я людей і довкілля.

Метою роботи є розробка та реалізація інформаційної системи для прогнозування забруднення повітря, що охоплює автоматизований збір даних з екологічних і метеорологічних джерел, обробку великих обсягів інформації, навчання моделей статистичного та машинного навчання при оцінці індексу якості повітря (AQI) та надання прогнозів у реальному часі.

Завданнями магістерської роботи було визначено:

1. Провести аналіз сучасних методів оцінки забруднення повітря та їх ефективності.
2. Розробити інформаційну інфраструктуру для збору, обробки та зберігання метеорологічних і екологічних даних.
3. Реалізувати аналітичні модулі для прогнозування рівнів забруднення повітря на основі зібраних даних.
4. Виконати аналіз даних, включаючи очищення, візуалізацію та кореляційний аналіз, для виявлення тенденцій забруднення.

5. Навчити і оцінити модель прогнозування забруднення повітря, використовуючи методи машинного навчання.

6. Розгорнути модель у середовищі Amazon SageMaker та отримати результати машинного навчання.

Об'єкт дослідження. Системи моніторингу та прогнозування забруднення повітря, а також їх взаємозв'язок з метеорологічними та екологічними даними, що впливають на якість повітря.

Предмет дослідження. Підходи, методи та інструменти статистичного та машинного навчання в аналізі даних оцінки рівнів забруднення повітря, а також інформаційні технології, що використовуються для реалізації аналітичних систем прогнозування.

В ході проведеного дослідження використовувались **методи**: загально-наукові (спостереження, аналізу, моделювання, декомпозиції систем, індукції), емпіричні (спостереження, кореляційно-регресійний аналіз, аналіз метрик, тестування).

Наукова новизна дослідження полягає в розробці комплексної інформаційної системи для моніторингу, аналізу та прогнозування забруднення повітря, яка інтегрує сучасні методи статистичного та машинного навчання, хмарні сервіси Amazon Web Services (AWS) та автоматизовані процеси збору даних. Запропонована система дозволяє здійснювати прогнозування індексу якості повітря (AQI) у реальному часі, враховуючи часову залежність і багатовимірність екологічних та метеорологічних даних.

Структура роботи. Структуру магістерської роботи складають: вступ, три розділи, висновки, список використаних джерел та додатки. У першому розділі систематизовано теоретичні основи прогнозування забруднення повітря, розглянуто існуючі методи та моделі, а також сучасні підходи до аналізу екологічних даних. У другому розділі розроблено інформаційну інфраструктуру для збору, обробки та зберігання даних, включаючи налаштування AWS Lambda для автоматизації цих процесів. У третьому розділі викладено результати аналізу метеорологічних та екологічних даних,

навчання моделі прогнозування індексу якості повітря (AQI) та її розгортання у середовищі Amazon SageMaker.

Практичне значення роботи полягає у створенні інформаційної системи, що забезпечує автоматизований моніторинг, аналіз та прогнозування забруднення повітря для 28 міст України. Результати дослідження можуть бути використані екологічними організаціями, державними установами та іншими зацікавленими сторонами для ухвалення рішень щодо управління якістю повітря.

Апробація результатів дослідження:

1. Вишневський А. В., Горяшин А. С. Інформаційна система для прогнозування забруднення повітря за допомогою аналізу метеорологічних та екологічних даних. Прикладні інформаційні технології: матеріали V Всеукраїнської науково-практичної конференції здобувачів, аспірантів та молодих вчених (м. Вінниця, 24 травня 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024. С. 148 - 150.

2. Вишневський А. В., Потапова Н. А. Методи аналізу даних в інформаційній системі прогнозування показників забруднення повітря. Прикладні аспекти сучасних міждисциплінарних досліджень: ІІІ Міжнародна науково-практична конференція (м. Вінниця, 1 листопада 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПРОГНОЗУВАННЯ ЗАБРУДНЕННЯ ПОВІТРЯ

1.1 Основні поняття та елементи системи прогнозування забруднення повітря

Забруднення повітря є однією з найбільш критичних екологічних проблем сучасності, що впливає на здоров'я людини, біорізноманіття та клімат Землі. За даними Всесвітньої організації здоров'я, мільйони людей щорічно помирають від проблем, пов'язаних зі забрудненим повітрям. Розуміння і прогнозування динаміки забруднюючих речовин має вирішальне значення для формування політик охорони здоров'я та екологічної безпеки.

Інформаційні системи грають ключову роль у зборі, обробці та аналізі екологічних даних, включаючи дані про забруднення повітря. Використання передових технологій дозволяє не тільки фіксувати поточний стан атмосфери, але й ефективно прогнозувати майбутні зміни. Це включає в себе аналіз великих даних зі станцій моніторингу, супутників, а також отримання прогностичних моделей з використанням машинного навчання та штучного інтелекту. Ці системи надають можливість своєчасно реагувати на екологічні загрози, оптимізувати управління міськими та промисловими процесами, знижуючи ризики для здоров'я населення та забруднення довкілля.

Забруднення повітря визначається як наявність однієї чи декількох речовин у атмосфері, які шкідливі для здоров'я людини, рослин або тварин, або які можуть завдати шкоди матеріалам, знизити видимість, або викликати неприємний запах. Природні джерела (наприклад, вулкани, лісові пожежі, пил з пустель) і антропогенні джерела (такі як промислові викиди, автотранспорт, енергетичні станції) є основними винуватцями у викидах забруднюючих речовин у повітря. Предметна область дослідження охоплює вивчення джерел забруднення, їх характеристик, а також методів вимірювання і контролю концентрацій забруднювачів. Важливим аспектом є також вплив

метеорологічних умов на розповсюдження та концентрацію забруднювачів у повітрі.

Забруднення повітря можна класифікувати за декількома категоріями:

- Природні джерела включають вулкани, які викидають сірководень, попіл, і металеві оксиди; лісові пожежі, які викидають дим і вуглекислий газ; пил з пустель, який може переноситися на великі відстані.
- Антропогенні джерела охоплюють викиди від автомобілів, які включають вуглекислий газ, оксиди азоту, вуглеводні; промислові викиди, що складаються з сірководню, аміаку, хлору, і важких металів; а також теплові електростанції, які викидають сірку діоксид і золу.

Розуміння цих джерел є критично важливим для визначення стратегій моніторингу та контролю забруднення повітря. Аналіз джерел забруднення допомагає вченим і політикам розробляти цільові заходи для зменшення впливу забруднювачів на екологію та здоров'я людини.

Забруднення повітря є однією з ключових екологічних проблем України, особливо в урбанізованих регіонах. Згідно з даними моніторингу, Київ, Дніпро, Харків та Львів часто перевищують допустимі рівні концентрацій PM2.5 та NO_2. Основними джерелами забруднення є:

- Промислові підприємства.
- Викиди автотранспорту.
- Теплові електростанції.

Економічні та екологічні наслідки

Вплив забруднення повітря охоплює:

- Здоров'я населення: підвищення ризику респіраторних захворювань, астми, серцевих нападів і навіть передчасної смертності.
- Економічні витрати: витрати на лікування, втрату продуктивності праці, компенсації за шкоду здоров'ю.
- Кліматичні зміни: збільшення концентрації CO_2 та інших парникових газів.

Дослідження підтверджують, що поліпшення якості повітря може

зменшити економічні втрати на десятки мільярдів гривень щорічно.

1.2 Аналіз існуючих методів прогнозування

Існують різні методи і техніки для прогнозування забруднення повітря, включаючи статистичні моделі, математичне моделювання та штучний інтелект. Переважно використовуються детерміністичні моделі, які враховують фізичні процеси розповсюдження забруднювачів, а також емпіричні моделі, що базуються на історичних даних. Слід проаналізувати переваги та недоліки цих методів, оцінюючи їх точність, вартість впровадження та здатність до інтеграції з іншими системами моніторингу.

Детерміністичні методи мають наступні переваги:

- Точність. Детерміністичні моделі можуть дуже точно моделювати фізичні процеси, якщо їм надано достатньо деталізовану інформацію про умови.
- Зрозумілість. Ці моделі базуються на фізичних законах, що робить їх зрозумілими для фахівців у цій галузі.

Недоліками даних моделей є:

- Високі вимоги до даних вимагають великої кількості точних вхідних даних, включаючи параметри забруднення, метеорологічні дані, географічні дані тощо.
- Обчислювальні витрати вимагають значних обчислювальних ресурсів для вирішення складних рівнянь.

Емпіричні моделі мають переваги:

- Гнучкість. Легко адаптуються під різні типи даних і умов.
- Менші обчислювальні витрати. Типово потребують менше обчислювальних ресурсів порівняно з детерміністичними моделями.

Недоліки:

- Залежність від даних. Якість прогнозів сильно залежить від якості історичних даних.

- Обмежена інтерпретація. Можуть виявляти статистичні зв’язки без зрозуміння причинно-наслідкових зв’язків.

Моделі на основі штучного інтелекту (ШІ) мають переваги:

- Висока точність. Можуть ефективно обробляти великі набори даних і виявляти складні патерни, що часто не видимі для традиційних методів.
- Адаптивність. Моделі ШІ можуть навчатися і адаптуватися до нових даних у реальному часі.

Недоліки:

- Чорний ящик: Рішення, згенеровані алгоритмами ШІ, часто буває важко інтерпретувати.
- Потреба в даних: Необхідність у великих обсягах даних для ефективного навчання.

Порівняльний аналіз даних методів включає характеристику таких параметрів як-от інтеграція, вартість впровадження, точність та надійність:

- Інтеграція. Хоча детерміністичні та емпіричні моделі можуть ефективно інтегруватися з іншими системами моніторингу, моделі ШІ можуть вимагати спеціальних інтерфейсів для інтеграції через їх складність та потребу в ресурсах.
- Вартість впровадження. Детерміністичні моделі часто є найдорожчими через потребу в деталізованих даних і обчислювальних ресурсах; емпіричні моделі та ШІ можуть бути більш бюджетними альтернативами.
- Точність та надійність. ШІ може забезпечити вищу точність і надійність у складних умовах, де традиційні моделі виявляються неефективними.

Кожен із цих методів вносить свій вклад у покращення розуміння та управління забрудненням повітря, і вибір методу залежить від конкретних умов і доступних ресурсів.

1.3 Сучасні підходи до аналізу даних забруднення повітря

Сучасні технології, такі як машинне навчання та нейронні мережі, відкривають нові можливості для покращення точності та оперативності прогнозів забруднення повітря. Використання цих технологій дозволяє обробляти великі обсяги даних і швидко адаптуватися до змін умов. Важливо розглянути конкретні приклади застосування машинного навчання для прогнозування, з урахуванням їх переваг, таких як підвищення точності та зниження помилок у короткострокових прогнозах.

Машинне навчання (МН) пропонує потужні інструменти для аналізу великих обсягів даних, здатні ідентифікувати складні закономірності та взаємозв'язки, які часто залишаються невидимими для традиційних аналітичних методів. Використання МН у сфері моніторингу забруднення повітря включає декілька ключових напрямків:

- Прогнозування рівнів забруднювачів. Моделі машинного навчання, можуть прогнозувати концентрації забруднювачів на основі історичних даних та поточних метеорологічних умов.
- Виявлення джерел забруднення. Методи кластеризації та розпізнавання образів допомагають ідентифікувати потенційні джерела забруднення, аналізуючи спатіальні та темпоральні шаблони забруднення.
- Оптимізація моніторингу. Алгоритми оптимізації можуть використовуватися для планування місць розміщення станцій моніторингу повітря для максимізації покриття та ефективності мережі моніторингу.

Інтеграція МН з Інтернетом речей (IoT) і великими даними відкриває нові можливості для систем моніторингу та прогнозування забруднення повітря в реальному часі. Ці системи можуть неперервно збирати дані з різноманітних джерел, таких як сенсори якості повітря, метеостанції, транспортні засоби та супутники, і швидко обробляти ці дані для надання точних і актуальних прогнозів. Основними перевагами даного підходу є:

- Покращення реагування на аварійні ситуації: Системи, що використовують дані в реальному часі, можуть швидко виявляти піки

забруднення і автоматично сповіщати відповідні служби та громадськість, що дозволяє ефективно реагувати на надзвичайні ситуації.

- Адаптивність та масштабованість: Системи, що базуються на МН, можуть адаптуватися до змін у міському середовищі або промислових процесах, що дозволяє з часом покращувати точність прогнозів та забезпечувати масштабування від локального до глобального моніторингу.

Ці сучасні підходи значно підвищують можливості у сфері моніторингу та управління забрудненням повітря, пропонуючи більш ефективні та відповідальні рішення для збереження здоров'я людей та довкілля.

Основи машинного навчання в прогнозуванні забруднення повітря становлять базові принципи машинного навчання, обґрунтування вибору методу, формули та математична модель, що використовуються для прогнозування забруднення повітря. Машинне навчання поділяється на три основні типи залежно від способу роботи з даними:

Навчання з учителем (Supervised Learning). За цим підходом алгоритм працює із розміченими даними, де кожен зразок має відповідну мітку (результат). Завдання полягає у побудові моделі, яка здатна передбачати мітки для нових зразків. Формально задача класифікації полягає у побудові функції $f(x)$, яка для заданого вектора ознак x передбачає клас y:

$$(x) = y, \quad y \in \{ C_1, C_2, \ \dots, C_k \} \qquad (1.1)$$

де C_k – множина класів.

Навчання без учителя (Unsupervised Learning). Використовується для роботи з нерозміченими даними, щоб виявляти приховані структури або групи в даних. Завданням є виявлення функції $g(x)$, яка описує структуру даних:

$$g(x) \rightarrow \text{кластерів або латентних змінних.} \qquad (1.2)$$

Навчання з підкріпленням (Reinforcement Learning). Використовується

для задач, де алгоритм взаємодіє із середовищем і отримує винагороди або покарання за свої дії. Задача формалізується як максимізація сумарної винагороди R:

$$\max \sum_{t=0}^{T} R_t , \qquad (1.3)$$

де R_t – винагорода на кроці t.

Обґрунтованість вибору навчання з учителем підкріплена тим, що виконуються наступні критерії:

- У задачі доступні розмічені дані, що дозволяє ефективно навчити модель.
- Завдання передбачає класифікацію рівнів забруднення, що найкраще відповідає методу навчання з учителем.
- Алгоритми навчання з учителем (наприклад, Random Forest Classifier) є порівняно простими у реалізації та забезпечують високу точність.

Математична модель класифікації Random Forest – це ансамблевий метод, який використовує множину дерев рішень для класифікації. Побудова математичної моделі Random Forest охоплює наступні складові:

1. Формалізація задачі. Для набору навчальних даних $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, де x_y – вектор ознак, а y_i – мітка класу $Random\ Forest$ створює ансамбль N дерев рішень $h_t(x)$:

$$H(x) = \text{majority\_vote}\{h_1(x), h_2(x), \dots, h_N(x)\} \qquad (1.4)$$

де $H(x)$ – фінальне передбачення класу.

2. Побудова дерева рішень. Кожне дерево будується за наступним алгоритмом:

- обирається випадкова підвибірка даних із поверненням (bootstrap sampling);

- у кожному вузлі дерева обирається оптимальна ознака x_j і поріг τ, які мінімізують імпурність Джині G:

$$G = 1 - \sum_{k=1}^{K} p_k^2, \quad (1.5)$$

де p_k – частота зразків, що належать до класу k у вузлі.

3. Агрегація результатів. Під час класифікації кожне дерево $h_t(x)$ передбачає клас, і фінальний результат визначається голосуванням більшості:

$$H(x) = \arg\max_k \sum_{t=1}^{N} 1(h_t(x) = k), \quad (1.6)$$

де 1 – індикаторна функція, яка дорівнює 1, якщо$h_t(x) = k$, інакше 0.

4. Важливість ознаки x_j обчислюється як середнє зменшення імпурності Джині:

$$I(x_j) = \frac{1}{N} \sum_{t=1}^{N} \Delta G_t(x_j), \quad (1.7)$$

де $\Delta G_t(x_j)$ – зменшення імпурності в дереві е, пов'язане з використанням ознаки x_j.

Гіперпараметрами Random Forest є:

1. Кількість дерев (n_estimators). Чим більше дерев, тим вища точність, але це збільшує час обчислень.

2. Максимальна глибина дерева (max_depth). Контролює складність моделі.

3. Кількість ознак для кожного вузла (max_features). Задає випадковість вибору ознак, що сприяє зменшенню кореляції між деревами.

Random Forest є ефективним методом для задач класифікації, таких як прогнозування рівнів забруднення повітря, завдяки своїй стійкості до шуму,

гнучкості та здатності обробляти великі обсяги даних. Оцінка важливості ознаки для Random Forest проводиться за формулою:

$$\text{Imp}(X_k) = \frac{1}{T}\sum_{t=1}^{T} \Delta G_t(X_k) \qquad (1.8)$$

де $\text{Imp}(X_k)$ – важливість ознаки X_k,
$\Delta G_t(X_k)$ – зменшення імпурності в дереві ttt, пов'язане з ознакою X_k,
T – кількість дерев у моделі.

Обраний метод, Random Forest Classifier, відзначається наступними перевагами:

1. Гнучкість. Підходить для роботи з великими обсягами даних та складними взаємозв'язками.

2. Стійкість до шуму. Висока точність навіть за наявності некоректних даних.

3. Інтерпретованість. Можливість визначення вагомості ознак, що впливають на результат.

Інші алгоритми мають наступні харакетристики:

- SVM. Вимагає багато обчислювальних ресурсів, що робить його менш ефективним для великих наборів даних.
- Нейронні мережі. Мають високу точність, але вимагають значних обсягів даних та складнішої інфраструктури.

Метрики забруднення повітря є ключовими елементами для аналізу якості повітря та визначення потенційного впливу на здоров'я і довкілля. Важливо глибше розглянути найбільш значущі показники, які використовуються в інформаційній системі.

AQI (Air Quality Index) – це числовий індикатор якості повітря, який дозволяє оцінювати його стан у зрозумілому для громадськості форматі. AQI приймає значення в діапазоні від 1 до 5:

- 1: Дуже добре – якість повітря оптимальна.

- 2: Добре – якість повітря прийнятна.
- 3: Задовільно – якість повітря може мати незначний вплив на чутливі групи.
- 4: Погано – якість повітря може негативно впливати на здоров'я населення.
- 5: Дуже погано – серйозний ризик для здоров'я.

Для розрахунку AQI в нашій роботі використано формулу:

$$AQI = \frac{I_{\text{high}} - I_{\text{low}}}{C_{\text{high}} - C_{\text{low}}}(C - C_{\text{low}}) + I_{\text{low}} \qquad (1.9)$$

де C – виміряна концентрація забруднювача,

C_{low} і C_{high} – межі концентрації для цього рівня AQI,

I_{low} і I_{high} – межі індексу для відповідного діапазону концентрацій (від 1 до 5).

PM2.5 – це дрібнодисперсні частки діаметром менше 2.5 мкм, що можуть проникати глибоко в легені і навіть у кровотік. Обчислюється за формулою:

$$R = \frac{C_{\text{PM2.5}} - C_{\text{threshold}}}{C_{\text{threshold}}} \times 100 \qquad (1.10)$$

де R – відносний ризик,

$C_{\text{PM2.5}}$ – концентрація PM2.5,

$C_{\text{threshold}}$ – безпечний рівень концентрації.

NO_2 є одним із найпоширеніших забруднювачів, який утворюється внаслідок спалювання викопного палива. Його високі концентрації впливають на розвиток астми та інших респіраторних захворювань. У місті рівень NO_2 часто є показником інтенсивності руху транспорту:

$$\text{Impact}_{\text{NO2}} = C_{\text{NO2}} \times \text{Exposure}_{\text{time}} \qquad (1.11)$$

де Impact_{NO2} – вплив на здоров'я,

C_{NO2}– концентрація NO_2,

Exposure_{time} – тривалість експозиції.

SO_2 (Діоксид сірки) виникає внаслідок спалювання викопного палива і є однією з головних причин кислотних дощів. Розрахунок його впливу часто базується на добовій середній концентрації:

$$E_{SO2} = \int_{t_1}^{t_2} C_{SO2}(t)\ dt \qquad (1.12)$$

де E_{SO2} – викид SO_2. – загальний вплив за період часу,

$C_{SO2}(t)$ – концентрація SO_2 в момент часу t.

Формула моделювання концентрації забруднювача в реальному часі:

$$\frac{dC}{dt} = E - D(C) + R(C) \qquad (1.13)$$

Де C – концентрація забруднювача,

E – емісія,

D(C) – процеси розсіювання,

R(C) – хімічні реакції.

Метрики забруднення часто взаємопов'язані через джерела походженняабо хімічні реакції в атмосфері. Так, підвищення рівня NO_2 може знижувати концентрацію O_3 через хімічну взаємодію. Аналіз цих зв'язків дозволяє моделювати складні екологічні процеси та визначати джерела забруднення.

Інформаційні системи дозволяють виконувати моделювання в реальному часі. Це передбачає інтеграцію сенсорів, які передають дані про якість повітря до центральної системи.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було систематизовано теоретичні основи прогнозування забруднення повітря, розглянуто існуючі методи і моделі, такі як детерміністичні, емпіричні та на основі машинного навчання. Проведений аналіз виявив переваги та недоліки кожного підходу, що дозволило визначити найбільш перспективні методи для задач прогнозування, зокрема використання штучного інтелекту. Окрему увагу приділено сучасним підходам, які базуються на аналізі великих даних і використанні хмарних технологій. Отримані результати створили міцну базу для подальшого розроблення інформаційної системи прогнозування забруднення повітря.

На основі аналізу хічних та екологічних складників досліджуваного процесу обрано метрики оцінювання процесу забруднення, які у подальшому мають стати основою для формалізації моделі машинного навчання для аналізу даних.

РОЗДІЛ 2
РОЗРОБКА ІНФОРМАЦІЙНОЇ ІНФРАСТРУКТУРИ ДЛЯ АНАЛІЗУ ДАННИХ В ІНФОРМАЦІЙНІЙ СИСТЕМІ

2.1 Створення інфраструктури для збору даних

Цей підрозділ визначає загальні цілі та структуру програми, що розробляється. Система має на меті збір, обробка, аналіз та візуалізація даних про забруднення повітря, щоб користувачі могли отримати інформацію про екологічний стан конкретних регіонів.

У цьому розділі детально описано покрокову реалізацію інформаційної системи для аналізу та прогнозування рівнів забруднення повітря з використанням сервісів Amazon Web Services (AWS). Буде розглянуто всі виконані дії, включаючи створення Lambda-функцій, налаштування S3-бакетів, обробку даних у SageMaker, розробку та розгортання моделі прогнозування. Код, використаний на кожному етапі, буде представлено та пояснено..

Розроблено Lambda-функцію на Python, яка здійснює періодичний збір даних про якість повітря з API для обраних міст України. Функція відправляє HTTP-запити до API, отримує дані та зберігає їх у S3-бакеті у форматі JSON. У консолі AWS Lambda обрано середовище виконання Python 3.11.

Код Lambda-функції:

```
import boto3
import requests
import json
import time
import os
def lambda_handler(event, context):
    s3 = boto3.client('s3')
    bucket = 'air-pollution-raw-data'
    api_key = os.getenv('API_KEY')
```

```
cities = [
    {‘city’: ‘Cherkasy’, ‘lat’: 49.4444, ‘lon’: 32.0598},
    # ... інші міста ...
    {‘city’: ‘Zhytomyr’, ‘lat’: 50.2547, ‘lon’: 28.6573}
]
for city_info in cities:
    city = city_info[‘city’]
    lat = city_info[‘lat’]
    lon = city_info[‘lon’]
    response = requests.get(
        ‘http://api.openweathermap.org/data/2.5/air_pollution’,
        params={‘lat’: lat, ‘lon’: lon, ‘appid’: api_key}
    )
    if response.status_code == 200:
        data = response.json()
        data[‘location’] = {‘city’: city, ‘lat’: lat, ‘lon’: lon}
        timestamp = int(time.time())
        datetime_obj = datetime.fromtimestamp(timestamp)
        date_path = datetime_obj.strftime(‘%Y/%m/%d’)
        file_name = f’air_pollution_data_{city}_{timestamp}.json’
        # Формуємо ключ з назвою міста та датою
        key = f"{city}/{date_path}/{file_name}"
        # Зберігаємо дані в S3 з оновленим ключем
        s3.put_object(
            Bucket=bucket,
            Key=key,
            Body=json.dumps(data)
        )
        print(f"Дані для міста {city} збережено як {key}")
```

```
    else:
        print(f"Помилка при отриманні даних для міста {city}: {response.status_code}")
```

Опис коду:

Імпорт бібліотек:

```
import boto3
import requests
import json
import time
import os
```

- boto3: AWS SDK для Python, який використовується для роботи з S3 (створення бакетів, завантаження файлів тощо).
- requests: Бібліотека для виконання HTTP-запитів до API, яка забезпечує простий спосіб взаємодії з веб-сервісами.
- json: Стандартна бібліотека Python для роботи з JSON-даними, що дозволяє серіалізувати і десеріалізувати дані.Функція lambda_handler. Основна функція, яка виконується при запуску Lambda.
- time: Використовується для отримання поточного часу в форматі UNIX-мітки.
- os: Забезпечує доступ до змінних середовища, включаючи API-ключ.

Функція lambda_handler:

```
def lambda_handler(event, context):
  s3 = boto3.client('s3')
  bucket = 'air-pollution-raw-data'
  api_key = os.getenv('API_KEY')
```

- s3 = boto3.client('s3'): Ініціалізація клієнта для взаємодії з S3.
- bucket = 'air-pollution-raw-data': Назва бакету, куди зберігатимуться зібрані дані.
- api_key = os.getenv('API_KEY'): Отримання API-ключа з змінної середовища для доступу до OpenWeatherMap API.

Список міст:

```
cities = [
    {‘city’: ‘Cherkasy’, ‘lat’: 49.4444, ‘lon’: 32.0598},
    # ... інші міста ...
    {‘city’: ‘Zhytomyr’, ‘lat’: 50.2547, ‘lon’: 28.6573}
]
```

- cities: Масив, що містить словники з інформацією про міста (назва міста, широта, довгота).
- Для кожного міста з цього списку виконується запит до API.

Запит до API та обробка відповіді:

```
response = requests.get(
    ‘http://api.openweathermap.org/data/2.5/air_pollution’,
    params={‘lat’: lat, ‘lon’: lon, ‘appid’: api_key}
)
```

- requests.get(): Виконує HTTP-запит до OpenWeatherMap API.
- params: Передача параметрів, включаючи координати міста (lat, lon) та API-ключ (appid).
- Перевірка відповіді: Перевіряється статус відповіді API за допомогою response.status_code.

Обробка успішної відповіді:

```
if response.status_code == 200:
    data = response.json()
    data[‘location’] = {‘city’: city, ‘lat’: lat, ‘lon’: lon}
```

- response.json(): Конвертує отримані дані у формат JSON.
- data[‘location’]: Додає до JSON дані про місце (назву міста та координати).

Формування структури файлу та зберігання в S3:

```
timestamp = int(time.time())
datetime_obj = datetime.fromtimestamp(timestamp)
```

```
date_path = datetime_obj.strftime('%Y/%m/%d')
file_name = f'air_pollution_data_{city}_{timestamp}.json'
key = f"{city}/{date_path}/{file_name}"
s3.put_object(
    Bucket=bucket,
    Key=key,
    Body=json.dumps(data)
)
```

- timestamp: Поточний час у форматі UNIX.
- datetime_obj.strftime('%Y/%m/%d'): Форматування дати для створення структури папок у S3.
- file_name: Унікальне ім'я файлу, що включає назву міста та мітку часу.
- key: Повний шлях до файлу в S3 (папки згруповані за містами та датою).
- s3.put_object(): Завантаження файлу у бакет S3.

Логування та обробка помилок:

```
print(f"Дані для міста {city} збережено як {key}")
else:
    print(f"Помилка при отриманні даних для міста {city}:
{response.status_code}")
```

- Успішне завантаження: Логування інформації про збережений файл.
- Помилки: У разі помилки логуються повідомлення про статус відповіді.

2.2 Розробка механізму обробки та зберігання даних

Цей підрозділ присвячений обробці зібраних даних та підготовці їх до аналізу. Створено другу Lambda-функцію, яка тригериться при додаванні

нового файлу в S3-бакет з сирими даними. Функція обробляє дані: витягує необхідні поля, очищує та перетворює їх у формат CSV, після чого зберігає в іншому S3-бакеті.

Код Lambda-функції:

```
import json
import boto3
from io import StringIO
import csv
import time
from datetime import datetime
def lambda_handler(event, context):
    s3 = boto3.client('s3')
    processed_bucket = 'air-pollution-processed-data'
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        print(f"Обробка файлу {key} з бакету {bucket}")
        # Завантаження даних з S3
        response = s3.get_object(Bucket=bucket, Key=key)
        content = response['Body'].read().decode('utf-8')
        data = json.loads(content)
        # Отримання інформації про місто та координати
        location = data.get('location', {})
        city = location.get('city')
        lat = location.get('lat')
        lon = location.get('lon')
        if not city or lat is None or lon is None:
          print("Інформація про місто або координати відсутня, пропускаємо файл.")
```

```
        continue
    processed_data = []
    for item in data['list']:
        row = {}
        row['city'] = city
        row['lat'] = lat
        row['lon'] = lon
        main = item.get('main', {})
        row['aqi'] = main.get('aqi')
        components = item.get('components', {})
        for k, v in components.items():
        row[k] = v
        row['dt'] = item.get('dt')
        processed_data.append(row)
    # Перетворення та очищення даних
    for row in processed_data:
        for key_, value in row.items():
            if key_ not in ['city', 'lat', 'lon', 'dt']:
                try:
                    row[key_] = float(value)
                except (ValueError, TypeError):
                    row[key_] = None
    # Видалення рядків з None значеннями
    processed_data = [
        row for row in processed_data
        if all(value is not None for value in row.values())
    ]
    if not processed_data:
        print("Немає валідних даних для збереження.")
        continue
```

```
        # Збереження оброблених даних в CSV
        fieldnames = processed_data[0].keys()
        csv_buffer = StringIO()
        writer = csv.DictWriter(csv_buffer, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(processed_data)
        # Формування шляху з датою для збереження
        timestamp = processed_data[0]['dt']
        datetime_obj = datetime.fromtimestamp(timestamp)
        date_path = datetime_obj.strftime('%Y/%m/%d')
        file_name = f"processed_data_{city}_{timestamp}.csv"
        processed_key = f"{city}/{date_path}/{file_name}"
        s3.put_object(
            Bucket=processed_bucket,
            Key=processed_key,
            Body=csv_buffer.getvalue()
        )
            print(f"Оброблені дані збережено як {processed_key} у бакеті
{processed_bucket}")
```

Опис коду:

Імпорт необхідних бібліотек:

```
import json
import boto3
from io import StringIO
import csv
import time
from datetime import datetime
```

- json: Для серіалізації та десеріалізації JSON-даних.
- boto3: AWS SDK для роботи з S3 (отримання та завантаження файлів).
- StringIO: Для створення віртуального текстового буфера, який

дозволяє зберігати CSV-дані в пам’яті перед завантаженням у S3.

- csv: Бібліотека для створення та маніпуляції CSV-файлами.
- time: Для роботи з часовими мітками UNIX.
- datetime: Для перетворення та форматування часу

Головна функція lambda_handler:

```
def lambda_handler(event, context):
    s3 = boto3.client('s3')
    processed_bucket = 'air-pollution-processed-data'
```

- event: Містить дані про подію, яка викликала функцію Lambda (наприклад, додавання нового файлу в S3-бакет).
- context: Надає інформацію про середовище виконання Lambda.
- s3 = boto3.client('s3'): Ініціалізація клієнта S3 для взаємодії з бакетами.
- processed_bucket: Назва бакету для збереження оброблених даних.

Обробка подій S3:

```
for record in event['Records']:
    bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']
    print(f"Обробка файлу {key} з бакету {bucket}")
```

- event['Records']: Містить перелік подій, пов’язаних із додаванням файлів у S3.
- bucket і key: Витягують назву бакету та ключ (шлях) файлу, який було додано.

Завантаження та читання JSON-файлу:

```
response = s3.get_object(Bucket=bucket, Key=key)
content = response['Body'].read().decode('utf-8')
data = json.loads(content)
```

- s3.get_object(): Отримує вміст файлу з вказаного бакету та ключа.
- response['Body'].read(): Читає вміст файлу як байти.
- decode('utf-8'): Конвертує байти у текст.

- json.loads(): Перетворює JSON-рядок у Python-словник.

Витягнення інформації про місце та перевірка полів:

```
location = data.get('location', {})
city = location.get('city')
lat = location.get('lat')
lon = location.get('lon')
if not city or lat is None or lon is None:
    print("Інформація про місто або координати відсутня, пропускаємо файл.")
    continue
```

- data.get(): Використовується для безпечного отримання полів JSON, навіть якщо вони відсутні.
- Перевірка: Якщо місто або координати відсутні, файл пропускається.

Формування оброблених даних:

```
processed_data = []
for item in data['list']:
    row = {
        'city': city,
        'lat': lat,
        'lon': lon,
        'aqi': item.get('main', {}).get('aqi'),
        'dt': item.get('dt')
    }
    components = item.get('components', {})
    for k, v in components.items():
        row[k] = v
    processed_data.append(row)
```

- data['list']: Містить масив з даними про забруднювачі.
- row: Створюється словник для кожного запису з базовими полями:
- Назва міста (city), координати (lat, lon), індекс якості повітря (aqi), часові мітки (dt).

- item.get('components'): Додає дані про концентрацію окремих забруднювачів.

Очищення та перетворення даних:

```
for row in processed_data:
    for key_, value in row.items():
        if key_ not in ['city', 'lat', 'lon', 'dt']:
            try:
                row[key_] = float(value)
            except (ValueError, TypeError):
                row[key_] = None
```

- Перетворення у float: Концентрації забруднювачів приводяться до числового формату.
- Обробка помилок: Некоректні значення замінюються на None.

Видалення некоректних рядків:

```
processed_data = [
    row for row in processed_data
    if all(value is not None for value in row.values())
]
```

- Видаляються всі рядки, які містять значення None.

Збереження у CSV:

```
fieldnames = processed_data[0].keys()
csv_buffer = StringIO()
writer = csv.DictWriter(csv_buffer, fieldnames=fieldnames)
writer.writeheader()
writer.writerows(processed_data)
```

- csv_buffer: Буфер у пам'яті для збереження CSV-файлу.
- DictWriter: Використовується для запису словників у CSV.

Збереження файлу у S3:

```
timestamp = processed_data[0]['dt']
datetime_obj = datetime.fromtimestamp(timestamp)
```

```
date_path = datetime_obj.strftime('%Y/%m/%d')
file_name = f"processed_data_{city}_{timestamp}.csv"
processed_key = f"{city}/{date_path}/{file_name}"
s3.put_object(
    Bucket=processed_bucket,
    Key=processed_key,
    Body=csv_buffer.getvalue()
)
```

- Формування шляху: Дата та місто використовуються для організації папок.
- s3.put_object(): Завантажує CSV-файл у бакет air-pollution-processed-data.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було розроблено інформаційну інфраструктуру для збору, обробки та зберігання екологічних даних. Інфраструктура включає автоматизований збір даних за допомогою AWS Lambda, їх зберігання у S3-бакетах, а також обробку для подальшого аналізу. Реалізовані процеси забезпечують надійність, масштабованість і зручність використання системи. Створена архітектура відповідає сучасним вимогам до систем екологічного моніторингу, дозволяючи автоматизувати рутинні процеси, мінімізувати людський фактор і підвищити точність обробки даних. Результати забезпечують основу для ефективного аналізу й прогнозування.

РОЗДІЛ 3
АНАЛІЗ ДАНИХ ДЛЯ ПРОГНОЗУВАННЯ ЗАБРУДНЕННЯ ПОВІТРЯ ЗА ДОПОМОГОЮ АНАЛІЗУ МЕТЕОРОЛОГІЧНИХ ТА ЕКОЛОГІЧНИХ ДАНИХ

3.1. Відтворення процесу попереднього завантаження даних в середовищі Amazon SageMaker

У цьому підрозділі описано процес дослідницького аналізу даних (EDA) з використанням SageMaker.

Створено ноутбук у SageMaker для аналізу та візуалізації даних. Встановлено необхідні бібліотеки для роботи з даними та побудови графіків.

Дії:

- Створення ноутбука:

1. У консолі SageMaker створено новий ноутбук з типом інстансу ml.t2.medium.

- Налаштування середовища:

1. Встановлено необхідні бібліотеки за допомогою команд:

```
!pip install pandas boto3 matplotlib seaborn folium
```

Підключення до S3:

- Використано бібліотеку boto3 для взаємодії з S3.

Дані попередньо завантажено з S3-бакету з обробленими даними. Виконано їх об’єднання, очищення від пропусків та дублікатів, а також перетворення часових міток у зручний формат.

Код для завантаження даних:

```
import boto3
import pandas as pd
from io import StringIO
s3 = boto3.client(‘s3’)
bucket = ‘air-pollution-processed-data’
```

```
def list_files_in_s3(bucket, prefix=‘‘):
    paginator = s3.get_paginator(‘list_objects_v2’)
    pages = paginator.paginate(Bucket=bucket, Prefix=prefix)
    files = []
    for page in pages:
        if ‘Contents’ in page:
            for obj in page[‘Contents’]:
                files.append(obj[‘Key’])
    return files
files = list_files_in_s3(bucket)
dataframes = []
for file in files:
    if file.endswith(‘.csv’):
        response = s3.get_object(Bucket=bucket, Key=file)
        content = response[‘Body’].read().decode(‘utf-8’)
        df = pd.read_csv(StringIO(content))
        dataframes.append(df)
full_df = pd.concat(dataframes, ignore_index=True)
```

Опис коду:

- Функція list_files_in_s3: Отримує список всіх файлів у бакеті.
- Завантаження та об’єднання даних:

1. Для кожного CSV-файлу завантажуються дані та додаються до списку DataFrame.

2. Використовується pd.concat для об’єднання всіх DataFrame в один.

Очищення та підготовка даних:

```
# Перевірка на пропущені значення
print(full_df.isnull().sum())
# Видалення рядків з пропущеними значеннями
clean_df = full_df.dropna()
# Видалення дублікатів
```

```
clean_df = clean_df.drop_duplicates()
# Перетворення часових міток у формат datetime
clean_df['datetime'] = pd.to_datetime(clean_df['dt'], unit='s')
# Видалення стовпця 'dt'
clean_df = clean_df.drop(columns=['dt'])
```

Опис коду:

- Перевірка та видалення пропусків та дублікатів:

1. Використовуються методи isnull(), dropna() та drop_duplicates().

- Перетворення часу:

1. Створюється новий стовпець datetime з використанням pd.to_datetime.

3.2 Аналіз часових рядів для виявлення тенденцій забруднення повітря

Проведено аналіз часових рядів для виявлення тенденцій забруднення повітря. Побудовано теплові карти для географічного аналізу розподілу забруднювачів. Виконано кореляційний аналіз між різними параметрами забруднення.

Аналіз часових рядів:

```
import matplotlib.pyplot as plt
# Встановлення індексу дати
clean_df.set_index('datetime', inplace=True)
# Фільтрація даних по місту Вінниця
kyiv_data = clean_df[clean_df['city'] == 'Vinnytsia']
# Побудова графіку
plt.figure(figsize=(12,6))
plt.plot(kyiv_data.index, kyiv_data['aqi'], label='AQI')
plt.title('Рівень AQI у Вінниці з часом')
plt.xlabel('Дата')
plt.ylabel('AQI')
```

```
plt.legend()
plt.show()
```

Опис коду:

- Фільтрація даних. Вибрано дані для міста Вінниця
- Побудова графіку. Використовується matplotlib для візуалізації змін AQI з часом.

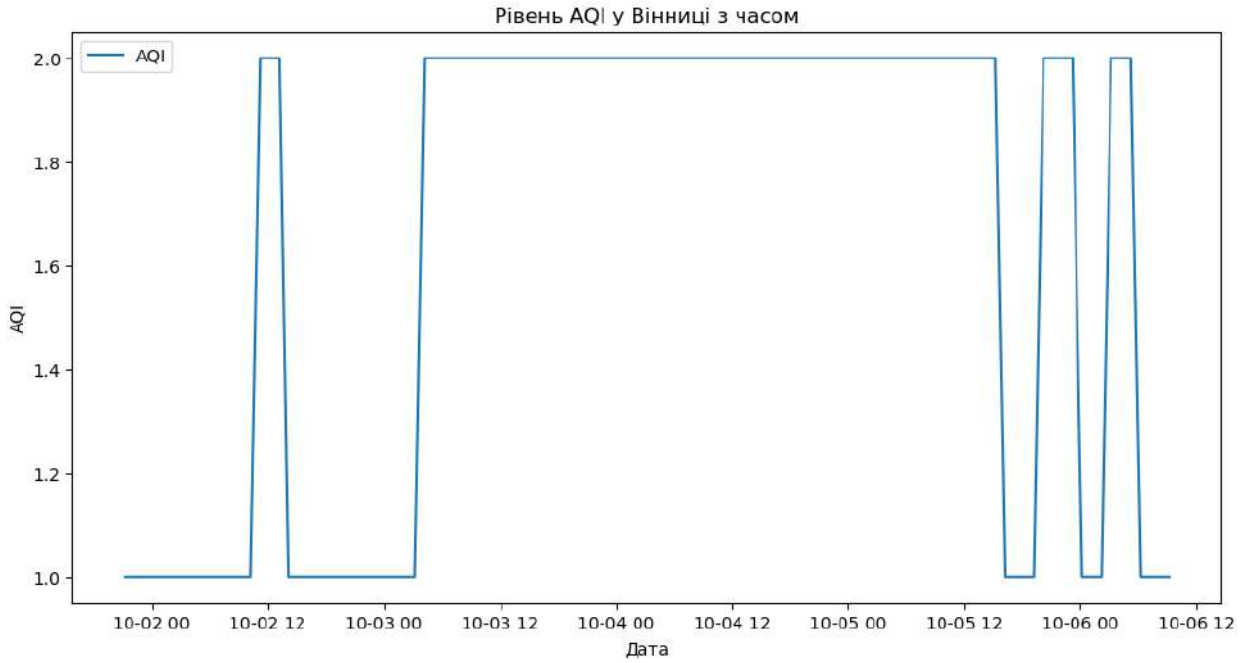

Рисунок 3. 1 – Приклад графіка AQI

Географічний аналіз:

```
import folium
from folium.plugins import HeatMap
# Створення базової карти
ukraine_map = folium.Map(location=[49.0, 32.0], zoom_start=6)
# Підготовка даних для теплової карти
heat_data = [[row['lat'], row['lon'], row['aqi']] for index, row in clean_df.iterrows()]
# Додавання теплової карти
HeatMap(heat_data).add_to(ukraine_map)
# Відображення карти
```

ukraine_map.save(‘ukraine_air_pollution_map.html’)

Опис коду:

- Створення карти з центром в Україні:

1. Використовується бібліотека folium.

- Додавання теплової карти:

1. Використовується HeatMap для візуалізації інтенсивності забруднення.

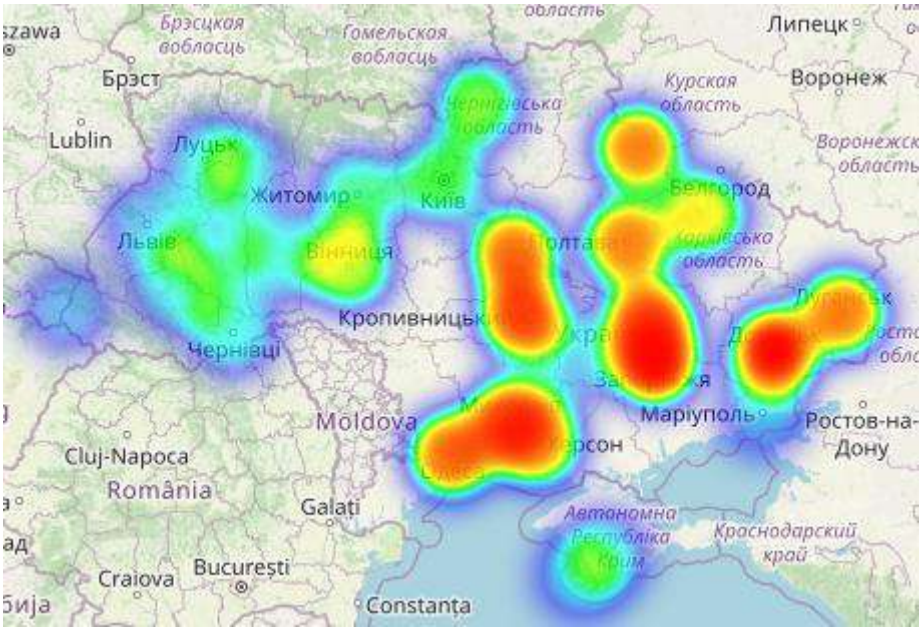

Рисунок 3.2. – Приклад теплової карти:

Кореляційний аналіз:

```
import seaborn as sns
# Вибір числових колонок
numeric_cols = clean_df.select_dtypes(include=[‘float64’, ‘int64’]).columns
# Обчислення кореляційної матриці
corr_matrix = clean_df[numeric_cols].corr()
# Відображення теплової карти кореляцій
plt.figure(figsize=(12,10))
sns.heatmap(corr_matrix, annot=True, cmap=‘coolwarm’)
plt.title(‘Кореляційна матриця’)
```

plt.show()

Опис коду:

- Обчислення кореляцій:

1. Використовується метод corr() для числових колонок.

- Візуалізація:

1. Використовується seaborn для відображення теплової карти кореляцій.

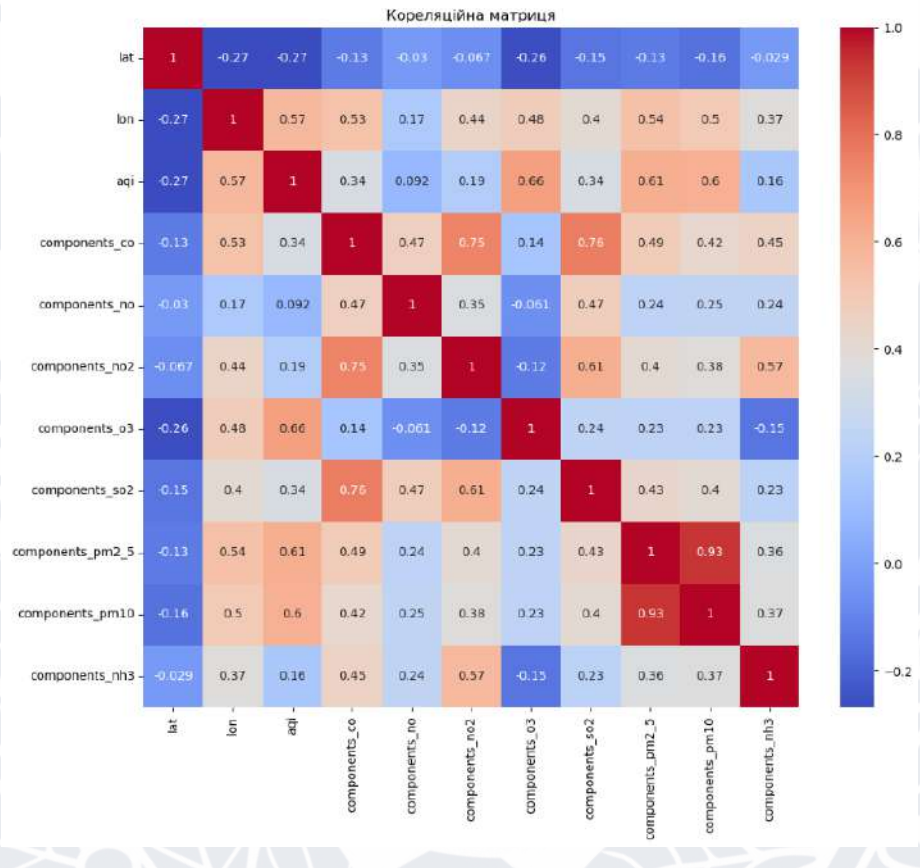

Рисунок 3. 3 – Приклад кореляційної матриці

Кореляційна матриця є важливим інструментом статистичного аналізу, що дозволяє виявляти взаємозв’язки між числовими змінними в наборі даних. У нашому дослідженні, яке стосується забруднення повітря, кореляційна матриця використовується для оцінки ступеня лінійного взаємозв’язку між різними забруднювачами, такими як PM2.5, NO_2, SO_2, CO та іншими компонентами, а також для розуміння їх впливу на індекс якості повітря (AQI). Це дає змогу не лише аналізувати взаємозалежність окремих змінних, а й

виявляти приховані тенденції, які можуть бути корисними для моделювання та прогнозування.

Загалом, кореляційна матриця представляє собою квадратну таблицю, де кожна комірка містить коефіцієнт кореляції між двома змінними. Значення коефіцієнта кореляції коливається від -1 до +1. ($r>0$) вказує на позитивну кореляцію між змінними, тобто коли одна змінна збільшується, інша також має тенденцію до зростання. Наприклад, високі рівні PM2.5 часто супроводжуються високими значеннями PM10, що свідчить про їх спільне джерело або подібні механізми утворення. Негативна кореляція ($r<0$) означає, що зростання однієї змінної супроводжується зниженням іншої, як це може спостерігатися між NO_2 та O_3 через хімічні реакції, які знижують концентрацію озону за підвищення рівня оксидів азоту. Кореляція близька до нуля ($r\approx0$) свідчить про відсутність лінійного зв'язку між змінними, що може означати незалежність їх джерел або впливів.

Для екологічного аналізу кореляційна матриця відіграє центральну роль у виявленні факторів, які найбільше впливають на якість повітря. Наприклад, високі значення кореляції між PM2.5 і AQI вказують на те, що концентрація дрібнодисперсних часток є ключовим показником загального забруднення. Це підтверджується численними дослідженнями, які демонструють, що PM2.5 має суттєвий вплив на здоров'я людини, проникаючи в легені та кровотік і спричиняючи серцево-судинні захворювання. У той же час, кореляція між SO_2 і AQI може бути менш вираженою у регіонах з переважно транспортними джерелами забруднення, де оксиди азоту відіграють значно більшу роль.

Кореляційна матриця також допомагає визначати проблему мультиколінеарності, яка виникає, коли дві або більше незалежні змінні в моделі мають високий ступінь кореляції між собою ($r>0.8$). Це може призвести до надмірної ваги однієї змінної в моделі машинного навчання та спотворення оцінки параметрів. Наприклад, висока кореляція між PM2.5 і PM10 може спричинити труднощі в інтерпретації моделі, оскільки ці змінні можуть взаємно підсилювати свій вплив. Для вирішення таких проблем

застосовуються різні підходи, зокрема виключення однієї з корельованих змінних, створення їх комбінації (наприклад, середнє значення), або використання регуляризаційних методів, таких як Lasso чи Ridge.

Окрім технічних аспектів, кореляційна матриця надає важливу інформацію для практичних екологічних заходів. Висока кореляція між певними забруднювачами може свідчити про їх спільні джерела. Наприклад, високі рівні NO_2 та CO часто вказують на інтенсивний транспортний рух у міських районах, що дозволяє ідентифікувати ці зони як пріоритетні для впровадження обмежень на викиди. У той же час, низька кореляція між SO_2 та CO може вказувати на те, що ці забруднювачі походять із різних джерел, таких як промислові підприємства та транспорт.

Аналіз кореляційної матриці є важливим інструментом для вибору ознак у моделюванні. Змінні, які мають сильну кореляцію з AQI ($r>0$), наприклад PM2.5, повинні бути включені до моделі, оскільки вони є ключовими факторами прогнозування. З іншого боку, змінні з низьким значенням кореляції можуть бути менш інформативними і незначно впливати на результати прогнозування.

Розглянемо конкретний приклад інтерпретації кореляційної матриці. Якщо коефіцієнт кореляції між PM2.5 і AQI становить 0.85, це вказує на сильний позитивний зв’язок, що свідчить про те, що зі збільшенням концентрації PM2.5 індекс якості повітря також погіршується. Такий зв’язок дозволяє прогнозувати рівень AQI, базуючись на вимірюваннях PM2.5. У випадку NO_2 та O_3 коефіцієнт кореляції може становити -0.65, що вказує на їх антагоністичну взаємодію. Це важливо враховувати, оскільки підвищення рівня оксидів азоту часто зменшує концентрацію озону через хімічні реакції. Нарешті, низька кореляція між CO та SO_2 свідчить про незалежність їх змін, що може бути результатом різних джерел або умов утворення.

Висновки з кореляційної матриці є багатогранними. Вони дозволяють не лише покращити точність моделі машинного навчання, включаючи найбільш

релевантні змінні та виключаючи мультиколінеарність, але й сприяють розробці ефективних стратегій зменшення забруднення.

Наприклад, знання про те, що PM2.5 є основним чинником підвищення AQI, може мотивувати до впровадження заходів для зниження рівня дрібнодисперсних часток, таких як контроль транспортних викидів чи модернізація промислового обладнання. Аналогічно, розуміння антагоністичної кореляції між NO_2 і O_3 може допомогти у виборі хімічних та технологічних засобів для контролю цих забруднювачів.

Таким чином, кореляційна матриця виступає потужним інструментом для виявлення та інтерпретації взаємозв'язків між компонентами забруднення повітря, сприяючи більш глибокому розумінню екологічних процесів і формуванню ефективних стратегій управління якістю повітря.

3.3 Використання Jupyter Labs для обробки та аналізу даних

Jupyter Labs – це інтерактивне середовище для створення та виконання коду, аналізу даних і документування досліджень. Воно є потужним інструментом для візуалізації результатів, завдяки інтеграції різних форматів, таких як код, текст, таблиці й графіки. У роботі Jupyter Labs надав гнучкість у виконанні обчислень, тестуванні моделей і презентації результатів аналізу даних. Основними перевагами для вибору Jupyter Labs є:

1. Інтерактивність: Можливість запускати код блоками ("cells"), що дозволяє одразу аналізувати результати кожного етапу.

2. Підтримка численних мов програмування: Хоча основною мовою є Python, Jupyter підтримує також R, Julia та інші мови.

3. Інтеграція з хмарними сервісами: Легко підключається до AWS для обробки великих обсягів даних.

4. Візуалізація: Інструмент інтегрується з бібліотеками, такими як Matplotlib, Seaborn та Folium, для створення високоякісних графіків і теплових карт.

Для роботи з Jupyter Labs було виконано наступні кроки:

1. Інтеграція з AWS:

- Налаштовано доступ до S3-бакетів для завантаження та зберігання даних через бібліотеку boto3.
- Виконувалось об'єднання даних із CSV-файлів, збережених у бакетах.

2. Основні бібліотеки. Для обробки та аналізу даних у Jupyter Labs було використано:

- pandas для роботи з таблицями.
- numpy для математичних обчислень.
- matplotlib і seaborn для візуалізації.
- folium для географічних аналізів.

Основними етапами роботи в Jupyter Labs:

1. Завантаження та попередня обробка даних: Було розроблено скрипти для завантаження даних із S3-бакетів, їх очищення та об'єднання в один DataFrame:

```
import boto3
import pandas as pd
from io import StringIO
# Підключення до S3
s3 = boto3.client('s3')
bucket_name = 'air-pollution-processed-data'
def load_data_from_s3(bucket, prefix):
    paginator = s3.get_paginator('list_objects_v2')
    pages = paginator.paginate(Bucket=bucket, Prefix=prefix)
    dataframes = []
    for page in pages:
        for obj in page['Contents']:
            response = s3.get_object(Bucket=bucket, Key=obj['Key'])
            df = pd.read_csv(StringIO(response['Body'].read().decode('utf-8')))
            dataframes.append(df)
    return pd.concat(dataframes, ignore_index=True)
```

```
data = load_data_from_s3(bucket_name, ‘‘)
data.dropna(inplace=True)
data[‘datetime’] = pd.to_datetime(data[‘dt’], unit=‘s’)
```

Аналіз часових рядів. У Jupyter Labs були створені графіки для аналізу змін індексу якості повітря (AQI) з часом.

```
import matplotlib.pyplot as plt
# Фільтрація даних по місту
kyiv_data = data[data[‘city’] == ‘Kyiv’]
# Візуалізація AQI з часом
plt.figure(figsize=(12, 6))
plt.plot(kyiv_data[‘datetime’], kyiv_data[‘aqi’], label=‘AQI’)
plt.title(‘Рівень AQI у Києві з часом’)
plt.xlabel(‘Дата’)
plt.ylabel(‘AQI’)
plt.legend()
plt.show()
```

Візуалізація географічного розподілу. Створено теплові карти для аналізу забруднення у різних регіонах.

```
import folium
from folium.plugins import HeatMap
ukraine_map = folium.Map(location=[49.0, 32.0], zoom_start=6)
heat_data = [[row[‘lat’], row[‘lon’], row[‘aqi’]] for index, row in data.iterrows()]
HeatMap(heat_data).add_to(ukraine_map)
ukraine_map.save(‘ukraine_air_pollution_map.html’)
```

Навчання моделей машинного навчання: Jupyter Labs використовувався для експериментів із навчанням моделей, таких як Random Forest, включаючи побудову лагових ознак:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```
# Створення лагових ознак
for lag in range(1, 25):
    data[f'aqi_lag_{lag}'] = data['aqi'].shift(lag)
data.dropna(inplace=True)
# Розділення на тренувальний і тестовий набори
X = data[[f'aqi_lag_{lag}' for lag in range(1, 25)]]
y = data['aqi']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)
# Навчання моделі
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Результати роботи в Jupyter Labs:

- Інтерактивність: Дозволило швидко тестувати та покращувати моделі машинного навчання.
- Візуалізація: Створені графіки й карти стали основою для інтерпретації результатів аналізу даних.
- Реплікабельність: Весь код і результати аналізу легко зберігаються у форматі ноутбуків (.ipynb), що спрощує відтворення досліджень.

Для оцінки якості моделі було використано метрику точності (accuracy), яка визначає відсоток правильно класифікованих прикладів у тестовій вибірці. Навчена модель Random Forest Classifier досягла точності 74.62%. Це свідчить про те, що модель може ефективно прогнозувати індекс якості повітря (AQI) на основі вхідних даних.

Одержане значення 75.62% вказує на достатньо високу точність моделі, враховуючи складність задачі та використані дані. Модель показала хорошу узагальнену здатність, однак потребує подальшого оптимального налаштування гіперпараметрів для покращення результатів.

Використання Jupyter Labs значно підвищило ефективність роботи над магістерським проєктом, дозволивши зручно інтегрувати всі етапи – від збору

та обробки даних до моделювання та візуалізації. Це середовище виявилося незамінним для роботи з великими наборами даних та демонстрації отриманих результатів.

Навчання та оцінка моделі прогнозування та опис процесу створення та навчання моделі для прогнозування рівнів забруднення повітря. Підготовлено навчальний та тестовий набори даних. Створено лагові ознаки для врахування часової залежності даних.

Код для підготовки даних:

```
# Вибір даних для міста Київ
city_data = clean_df[clean_df['city'] == 'Vinnytsia'].copy()
# Сортування за датою
city_data.sort_index(inplace=True)
# Створення лагових ознак
for lag in range(1, 25):  # Лаги від 1 до 24 годин
    city_data[f'aqi_lag_{lag}'] = city_data['aqi'].shift(lag)
# Видалення пропущених значень
city_data = city_data.dropna()
# Розділення на ознаки та цільову змінну
X = city_data[[f'aqi_lag_{lag}' for lag in range(1, 25)]]
y = city_data['aqi']
# Розділення на тренувальну та тестову вибірки
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)
```

Опис коду:

- Створення лагових ознак. Додаються нові стовпці з попередніми значеннями AQI.
- Видалення пропусків. Після зсуву з'являються NaN, які видаляються.
- Розділення даних. Дані розділяються на ознаки X та цільову змінну y.

Обрано модель випадкового лісу регресії (Random Forest Classifier). Модель навчено на тренувальних даних та оцінено її якість на тестовій вибірці

за допомогою метрик MAE, RMSE та R^2.

Навчання моделі:

```
from sklearn.ensemble import RandomForestClassifier
# Створення та навчання моделі
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Оцінка моделі:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
# Прогнозування на тестовій вибірці
y_pred = model.predict(X_test)
# Обчислення метрик
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f'MAE: {mae}')
print(f'RMSE: {rmse}')
print(f'R^2: {r2}')
# Візуалізація результатів
plt.figure(figsize=(12,6))
plt.plot(y_test.index, y_test, label='Фактичні')
plt.plot(y_test.index, y_pred, label='Прогнозовані')
plt.title('Порівняння фактичних та прогнозованих значень AQI')
plt.xlabel('Дата')
plt.ylabel('AQI')
plt.legend()
plt.show()
```

Опис коду:

- Навчання моделі: Використовується випадковий ліс з 100 деревами.
- Прогнозування та оцінка:

1. Виконується прогноз на тестовій вибірці.
2. Обчислюються метрики MAE, RMSE та R^2 для оцінки точності моделі.

- Візуалізація: Графік порівняння фактичних та прогнозованих значень AQI.

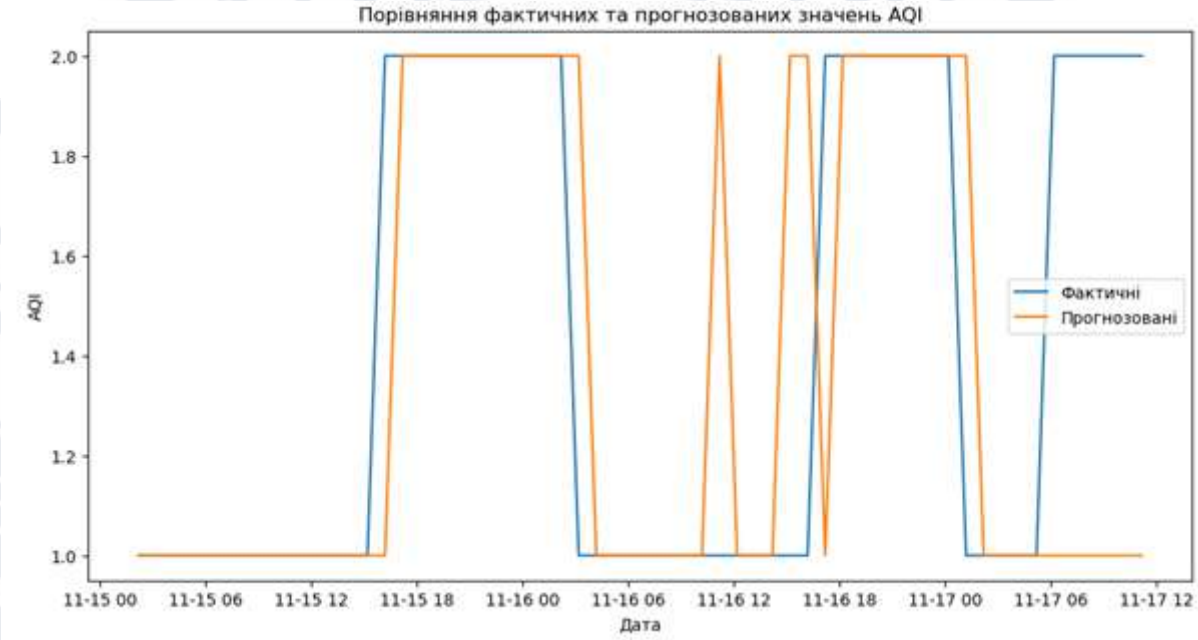

Рисунок 3.4 – Порівняння фактичних та прогнозованих значень AQI

Розглянемо метрики оцінки моделей MAE, RMSE та R^2.

Метрики MAE (Mean Absolute Error), RMSE (Root Mean Squared Error) та R^2 (R-squared) є стандартними показниками для оцінки якості прогнозування моделей машинного навчання. Вони допомагають зрозуміти, наскільки добре модель відображає залежності між вхідними даними та цільовими значеннями.

1. MAE (Mean Absolute Error). MAE визначає середню абсолютну різницю між фактичними та прогнозованими значеннями. Ця метрика показує, наскільки в середньому модель помиляється на кожному прогнозі та обчислюється за формулою:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \qquad (3.1)$$

де y_i – фактичне значення,

$\widehat{y_i}$ – прогнозоване значення,

n – кількість спостережень.

Особливості MAE:

- Легко інтерпретується: показує помилку в одиницях цільової змінної.
- Не так сильно реагує на великі помилки, як RMSE.

2. RMSE (Root Mean Squared Error). RMSE визначає корінь середнього квадрату різниць між фактичними та прогнозованими значеннями. Ця метрика підкреслює великі відхилення, адже квадратичне значення помилки збільшує вагу великих помилок. Формула для розрахунку має вигляд:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2} \qquad (3.2)$$

Особливості RMSE:

- Більш чутливий до великих помилок у порівнянні з MAE.
- Використовується, коли важливо мінімізувати великі відхилення.

3. R^2 (R-squared). R^2, або коефіцієнт детермінації, показує, яка частка варіації цільової змінної пояснюється моделлю. Ця метрика варіюється від 0 до 1, де 1 означає ідеальне передбачення, а 0 \u2014 модель не пояснює жодної варіації. Формула для розрахунку має вигляд:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \qquad (3.3)$$

де y_i фактичне значення,

$\widehat{y_i}$ прогнозоване значення,

$\bar{y}$ середнє значення фактичних даних.

Особливості R^2:

- Чим ближче до 1, тим краща модель.
- Якщо значення негативне, це означає, що модель гірша, ніж просте

середнє.

Розгортання та тестування моделі у середовищі SageMaker та опис процесу розгортання навченої моделі для використання в реальному часі. Модель збережено у форматі joblib та завантажено у S3-бакет для подальшого розгортання.

Збереження моделі та завантаження в S3:

```
import joblib
# Зберігаємо модель у файл
joblib.dump(model, ‘aqi_rf_model.joblib’)
# Завантажуємо модель у S3 для розгортання
s3 = boto3.client(‘s3’)
s3.upload_file(‘aqi_rf_model.joblib’,’ml-sage-bucket’,
‘models/aqi_rf_model.joblib’)
```

Опис коду:

- Збереження моделі: Модель зберігається у форматі joblib для подальшого використання.
- Завантаження в S3: Файл моделі завантажується у S3-бакет для доступу з SageMaker.

Інший блок:

```
import tarfile
model_file = ‘aqi_rf_model.joblib’
archive_name = ‘aqi_rf_model.tar.gz’
# Створюємо архів tar.gz, що містить модельний файл
with tarfile.open(archive_name, "w:gz") as tar:
    tar.add(model_file, arcname=model_file)
print(f"Архів {archive_name} створено успішно.")
```

Інший блок:

```
import boto3
s3 = boto3.client(‘s3’)
bucket_name = ‘ml-sage-bucket’
```

```
object_key = ‘models/aqi_rf_model.tar.gz’
file_path = ‘aqi_rf_model.tar.gz’
# Завантажуємо архів до S3
s3.upload_file(file_path, bucket_name, object_key)
print(f"Архів {file_path} завантажено до s3://{bucket_name}/{object_key}.")
```

За допомогою SageMaker розгорнуто модель як ендпоінт, що дозволяє виконувати прогнозування через API. Написано скрипт inference.py для обробки запитів та відповідей.

Написання скрипту inference.py для обробки запитів:

```
import json
import joblib
import os
import numpy as np
def model_fn(model_dir):
    # Завантаження моделі
    model = joblib.load(os.path.join(model_dir, ‘aqi_rf_model.joblib’))
    return model
def input_fn(request_body, request_content_type):
    if request_content_type == ‘application/json’:
        input_data = json.loads(request_body)
        return np.array(input_data[‘features’]).reshape(1, -1)
    else:
      raise ValueError("Unsupported content type: {}".format(request_content_type))
def predict_fn(input_data, model):
    prediction = model.predict(input_data)
    return prediction
def output_fn(prediction, response_content_type):
    if response_content_type == ‘application/json’:
        result = {‘prediction’: prediction.tolist()}
        return json.dumps(result)
```

```
    else:
        raise ValueError("Unsupported content type: {}".format(response_content_type))
```

Опис коду:

- Функція model_fn: Завантажує модель з директорії.
- Функція input_fn: Обробляє вхідні запити, перетворюючи JSON у numpy масив.
- Функція predict_fn: Виконує прогнозування з використанням моделі.
- Функція output_fn: Форматує результат у JSON для відповіді.

Розгортання моделі в SageMaker:

```
import boto3
import sagemaker
from sagemaker.sklearn.model import SKLearnModel
role = sagemaker.get_execution_role()
sagemaker_session = sagemaker.Session()
model = SKLearnModel(
    model_data='s3://ml-sage-bucket/models/aqi_rf_model.tar.gz',
    role=role,
    entry_point='inference.py',
    framework_version='1.2-1'
)
endpoint_name = 'aqi-prediction-endpoint'
predictor = model.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.large',
    endpoint_name=endpoint_name
)
```

Опис коду:

- Створення об'єкта моделі: Вказуються шлях до моделі в S3, роль IAM, скрипт inference.py та версія фреймворку.

- Розгортання моделі: Використовується метод deploy для створення ендпоінта з заданими параметрами.

Виконано тестові запити до ендпоінта для перевірки коректності роботи моделі. Отримані результати співставлено з очікуваними.

Код для тестування:

```
from sagemaker.predictor import Predictor
import json
# Підготовка тестових даних
test_features = X_test.iloc[0].values.tolist()
# Створення предиктора
predictor = Predictor(endpoint_name=endpoint_name,
sagemaker_session=sagemaker_session)
# Надсилання запиту
response = predictor.predict(
    data=json.dumps({'features': test_features}),
    initial_args={'ContentType': 'application/json'}
)
# Отримання прогнозу
prediction = json.loads(response)
print('Прогнозоване значення AQI:', prediction['prediction'][0])
```

Опис коду:

- Підготовка даних: Вибрані тестові ознаки для відправки на ендпоінт.
- Створення предиктора: Використовується клас Predictor для взаємодії з ендпоінтом.
- Виконання запиту та отримання відповіді: Надсилається запит з даними, отримується прогноз. Прогнозоване значення AQI: 1.386392857142857.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі проведено детальний аналіз екологічних та метеорологічних даних для прогнозування забруднення повітря.

Проведено детальний аналіз даних. Виконано очищення, візуалізацію та дослідження часових рядів для виявлення закономірностей у забрудненні повітря. Використовуючи середовище Amazon SageMaker, було виконано попереднє завантаження, очищення та аналіз метеорологічних і екологічних даних. Проведено навчання та оцінку моделі прогнозування, що підтвердило її працездатність через використання метрик, таких як MAE, RMSE та R^2.

Навчено модель Random Forest для прогнозування індексу якості повітря (AQI), яка показала високі результати за метриками точності. Розгорнута модель інтегрована у середовище Amazon SageMaker, що дозволило створити RESTful API для доступу до прогнозів у реальному часі. Результати підтверджують доцільність використання запропонованої системи для моніторингу та прогнозування забруднення повітря, забезпечуючи користувачів актуальною інформацією для прийняття рішень.

ВИСНОВКИ

У процесі виконання роботи було розроблено інформаційну систему для моніторингу, аналізу та прогнозування рівнів забруднення повітря, яка базується на використанні сервісів Amazon Web Services (AWS). Ця система вирішує завдання автоматичного збору даних, їх обробки та аналізу, що дозволяє отримувати актуальну інформацію про якість повітря в різних регіонах України. Її реалізація охоплює інтеграцію сучасних технологій хмарних обчислень, що забезпечує стабільність і масштабованість системи. За результатами виконаної роботи були отримані наступні висновки:

1. Проведено аналіз теоретичних основ вимірювання забруднення повітря та обрано метрики для подальшого використання у машинному навчанні.

2. Система дозволила автоматизувати всі етапи роботи з даними: від їх збору до отримання прогнозів у реальному часі. Дані про забруднення повітря збираються з погодинною регулярністю для 28 міст України, зберігаються у структурованому вигляді, що дозволяє з легкістю проводити аналіз за географічними та часовими параметрами. Оброблені дані зберігаються у форматі CSV, що забезпечує зручність для подальшого аналізу і побудови моделей. Процес збору та обробки даних визначено результатами: розроблено дві AWS Lambda-функції для автоматичного збору даних про забруднення повітря з API та їх обробки; дані збираються для 28 міст України з регулярністю кожну годину та зберігаються у структурованому вигляді в S3-бакетах; налаштовано автоматичний запуск функцій за допомогою Amazon EventBridge та тригерів S3.

3. Процес створення інфраструктури зберігання даних обумовлений проведенням етапів: організовано два S3-бакети для зберігання сирих та оброблених даних зі структурою за містами та датами; забезпечено необхідні права доступу та безпеку даних через налаштування IAM-ролей та політик.

4. Аналіз даних за допомогою Amazon SageMaker включав етапи: завантаження та об’єднання даних з S3 для проведення дослідницького аналізу; очищення даних, обробку пропусків та дублікатів, перетворення часових міток; проведення аналізу часових рядів, географічного аналізу розподілу забруднювачів та кореляційний аналіз між різними параметрами забруднення.

5. Розробка та навчання моделі прогнозування будувалась на: підготовці даних для моделювання, створенню лагових ознак для врахування часової залежності; навчанню моделі (Random Forest Classifier) для прогнозування індексу якості повітря (AQI); оцінці моделі за допомогою метрик MAE, RMSE та R^2, що підтвердило її придатність для прогнозування. У процесі аналізу даних виконано очищення, обробку пропусків, перетворення часових міток та видалення дублікатів, що дозволило створити якісний набір даних для моделювання. Проведений аналіз виявив ключові закономірності у розподілі забруднювачів та їх впливі на індекс якості повітря (AQI). Зокрема, було виявлено сильну кореляцію між PM2.5 та AQI, що підтверджує значення дрібнодисперсних часток як основного фактора забруднення повітря. Географічний аналіз показав, що густонаселені міста, такі як Київ та Харків, мають вищі рівні забруднення порівняно з менш урбанізованими регіонами.

6. Процес розгортання моделі та створення API будувався на: розгортанні моделі у середовищі AWS SageMaker, створенню ендпоінту для доступу до неї; розробці RESTful API за допомогою AWS API Gateway, що дозволяє взаємодіяти з моделлю через мережу; забезпеченні інтеграції моделі в інформаційну систему, що надає можливість виконувати прогнозування в реальному часі.

7. Розроблена модель прогнозування на основі Random Forest Classifier була навчена на підготовлених даних, що включали лагові ознаки для врахування часових залежностей. Зважаючи на обмежені ресурси, доступні під час навчання, модель показала прийнятні результати, що були оцінені за метриками MAE, RMSE та R^2. Зокрема, результати свідчать про здатність

моделі відображати взаємозв’язки між вхідними змінними та індексом AQI. Це підтверджує її придатність для використання у задачах прогнозування.

8. Розгорнута модель інтегрована у середовище AWS SageMaker, що забезпечило можливість створення API для отримання прогнозів у реальному часі. API дозволяє легко інтегрувати модель у зовнішні додатки або системи, забезпечуючи доступ до прогнозів для широкого кола користувачів.

9. Розроблена інформаційна система дозволяє виконувати моніторинг, аналіз і прогнозування забруднення повітря в автоматичному режимі. Вона забезпечує доступ до актуальної інформації про якість повітря, що може бути використано як державними установами, так і іншими зацікавленими сторонами для прийняття рішень щодо управління станом довкілля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Hunter, J. D. (n.d.). Matplotlib Documentation. URL: https://matplotlib.org/

2. McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

3. IEEE Xplore. (n.d.). IEEE Xplore Digital Library. URL: https://ieeexplore.ieee.org/

4. Brownlee, J. (2016). Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models and work Projects End-to-End. Machine Learning Mastery.

5. Python Software Foundation. Python Documentation. URL: https://www.python.org/doc/

6. Waskom, M. (n.d.). Seaborn Documentation. URL: https://seaborn.pydata.org/

7. Stack Overflow Community. (n.d.). Stack Overflow. URL: https://stackoverflow.com/

8. National Library of Medicine. (n.d.). PubMed. URL: https://pubmed.ncbi.nlm.nih.gov/

9. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

10. Waskom, M. L. (2021). Seaborn: Statistical Data Visualization. URL: https://seaborn.pydata.org/

11. Levy, A. (n.d.). Folium: Python Data. Visualizaion. URL: https://python-visualization.github.io/folium/

12. Streamlit Inc. (n.d.). Streamlit Documentation. URL: https://docs.streamlit.io/

13. Geron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.

14. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: https://www.deeplearningbook.org/

15. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson.

16. Amazon Web Services, Inc. (n.d.). Amazon SageMaker Developer Guide. URL: https://docs.aws.amazon.com/sagemaker/

17. Jupyter Documentation. (n.d.). Jupyter Notebook Documentation. URL: https://jupyter.org/documentation

18. Scipy Documentation. (n.d.). SciPy Documentation. URL: https://docs.scipy.org/doc/scipy/

19. Bokeh Contributors. (n.d.). Bokeh: Interactive Visualization for Modern Web Browsers. Retrieved from https://docs.bokeh.org/en/latest/

20. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. URL: https://scikit-learn.org/

21. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springe

22. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press.

23. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

24. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

25. Python Software Foundation. (n.d.). Python Documentation. URL: https://docs.python.org/

26. Pandas Developers. (n.d.). Pandas Documentation. URL: https://pandas.pydata.org/docs/

27. Pal, S. K., & Shiu, S. C. K. (2004). Foundations of Soft Case-Based Reasoning. Wiley-Interscience.

28. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

29. Van der Aalst, W. M. P. (2016). Process Mining: Data Science in Action. Springer.

30. Chollet, F. (2018). Deep Learning with Python. Manning Publications.

31. Webb, A. R., & Copsey, K. D. (2011). Statistical Pattern Recognition. Wiley.

32. Amazon Web Services (AWS). (n.d.). Amazon SageMaker Documentation. URL: https://docs.aws.amazon.com/sagemaker/

33. Scikit-learn Developers. (n.d.). Scikit-learn User Guide. URL: https://scikit-learn.org/stable/

34. NumPy Developers. (n.d.). NumPy Documentation. URL: https://numpy.org/doc/

35. Matplotlib Developers. (n.d.). Matplotlib Documentation. URL: https://matplotlib.org/stable/contents.html

36. Seaborn Developers. (n.d.). Seaborn Documentation. URL: https://seaborn.pydata.org/

37. OpenWeatherMap. (n.d.). OpenWeatherMap API Documentation. URL: https://openweathermap.org/api

38. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. DOI: 10.1023/A:1010933404324

39. Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesús, O. (2014). Neural Network Design. Martin Hagan Publishing.

40. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: https://www.deeplearningbook.org/

ДОДАТКИ

Додаток А

```
import boto3
import requests
import json
import time
import os
from datetime import datetime

def lambda_handler(event, context):
    s3 = boto3.client('s3')
    bucket = 'air-pollution-raw-data'
    api_key = os.getenv('API_KEY')
    cities = [
        {'city': 'Cherkasy', 'lat': 49.4444, 'lon': 32.0598},
        {'city': 'Chernihiv', 'lat': 51.4982, 'lon': 31.2893},
        {'city': 'Chernivtsi', 'lat': 48.2908, 'lon': 25.9345},
        {'city': 'Dnipro', 'lat': 48.4647, 'lon': 35.0462},
        {'city': 'Donetsk', 'lat': 48.0159, 'lon': 37.8028},
        {'city': 'Ivano-Frankivsk', 'lat': 48.9226, 'lon': 24.7111},
        {'city': 'Kharkiv', 'lat': 49.9935, 'lon': 36.2304},
        {'city': 'Kherson', 'lat': 46.6354, 'lon': 32.6169},
        {'city': 'Khmelnytskyi', 'lat': 49.4229, 'lon': 26.9871},
        {'city': 'Kropyvnytskyi', 'lat': 48.5079, 'lon': 32.2623},
        {'city': 'KryvyiRih', 'lat': 47.9102, 'lon': 33.3917},
        {'city': 'Kyiv', 'lat': 50.4501, 'lon': 30.4934},
        {'city': 'Luhansk', 'lat': 48.5740, 'lon': 39.3078},
        {'city': 'Lviv', 'lat': 49.8397, 'lon': 24.0297},
```

Продовження додатку А

```
    {‘city’: ‘Mariupol’, ‘lat’: 47.0957, ‘lon’: 37.5499},
    {‘city’: ‘Mykolaiv’, ‘lat’: 46.9750, ‘lon’: 31.9946},
    {‘city’: ‘Odesa’, ‘lat’: 46.4825, ‘lon’: 30.7233},
    {‘city’: ‘Poltava’, ‘lat’: 49.5883, ‘lon’: 34.5514},
    {‘city’: ‘Rivne’, ‘lat’: 50.6199, ‘lon’: 26.2516},
    {‘city’: ‘Sevastopol’, ‘lat’: 44.6054, ‘lon’: 33.5220},
    {‘city’: ‘Simferopol’, ‘lat’: 44.9521, ‘lon’: 34.1024},
    {‘city’: ‘Sumy’, ‘lat’: 50.9077, ‘lon’: 34.7981},
    {‘city’: ‘Ternopil’, ‘lat’: 49.5535, ‘lon’: 25.5948},
    {‘city’: ‘Vinnytsia’, ‘lat’: 49.2331, ‘lon’: 28.4682},
    {‘city’: ‘Lutsk’, ‘lat’: 50.7472, ‘lon’: 25.3254},
    {‘city’: ‘Uzhhorod’, ‘lat’: 48.6208, ‘lon’: 22.2879},
    {‘city’: ‘Zaporizhzhia’, ‘lat’: 47.8388, ‘lon’: 35.1396},
    {‘city’: ‘Zhytomyr’, ‘lat’: 50.2547, ‘lon’: 28.6573}
]
for city_info in cities:
    city = city_info[‘city’]
    lat = city_info[‘lat’]
    lon = city_info[‘lon’]
    response = requests.get(
        ‘http://api.openweathermap.org/data/2.5/air_pollution’,
        params={‘lat’: lat, ‘lon’: lon, ‘appid’: api_key}
    )

    if response.status_code == 200:
        data = response.json()
```

Продовження додатку А

```
        data['location'] = {'city': city, 'lat': lat, 'lon': lon}

        # Отримуємо поточний timestamp до години включно
        timestamp = datetime.now().strftime('%Y_%m_%d_%H')
        file_name = f'air_pollution_data_{city}_{timestamp}.json'

        # Формуємо ключ з назвою міста як префіксом
        key = f"{city}/{file_name}"

        # Зберігаємо дані в S3 з оновленим ключем
        s3.put_object(
            Bucket=bucket,
            Key=key,
            Body=json.dumps(data)
        )
        print(f"Дані для міста {city} збережено як {key}")
    else:
        print(f"Помилка при отриманні даних для міста {city}:
{response.status_code}")
```

Продовження додатку А

```
import json
import boto3
from io import StringIO
import csv
from datetime import datetime

def lambda_handler(event, context):
    s3 = boto3.client('s3')
    processed_bucket = 'air-pollution-processed-data'

    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        print(f"Обробка файлу {key} з бакету {bucket}")

        # Завантажуємо дані з S3
        response = s3.get_object(Bucket=bucket, Key=key)
        content = response['Body'].read().decode('utf-8')
        data = json.loads(content)

        # Отримуємо інформацію про місто та координати
        location = data.get('location', {})
        city = location.get('city')
        lat = location.get('lat')
        lon = location.get('lon')

        if not city or lat is None or lon is None:
```

Продовження додатку А

```
        print("Інформація про місто або координати відсутня, пропускаємо файл.")
        continue

    processed_data = []

    for item in data['list']:
        row = {}
        row['city'] = city
        row['lat'] = lat
        row['lon'] = lon

        main = item.get('main', {})
        row['aqi'] = main.get('aqi')

        components = item.get('components', {})
        for k, v in components.items():
            row[f'components_{k}'] = v

        row['dt'] = item.get('dt')

        processed_data.append(row)

    # Перетворення та очищення даних
    for row in processed_data:
        for key_, value in row.items():
            if key_ not in ['city', 'lat', 'lon', 'dt']:
```

Продовження додатку А

```
            try:
                row[key_] = float(value)
            except (ValueError, TypeError):
                row[key_] = None
# Видалення рядків з None значеннями
processed_data = [
    row for row in processed_data
    if all(value is not None for value in row.values())
]
if not processed_data:
    print("Немає валідних даних для збереження.")
    continue

# Збереження оброблених даних в CSV
fieldnames = processed_data[0].keys()
csv_buffer = StringIO()
writer = csv.DictWriter(csv_buffer, fieldnames=fieldnames)
writer.writeheader()
writer.writerows(processed_data)

# Формуємо ім’я файлу та ключ з назвою міста як префіксом
timestamp = datetime.now().strftime(‘%Y_%m_%d_%H’)
file_name = f"processed_data_{city}_{timestamp}.csv"
processed_key = f"{city}/{file_name}"

s3.put_object(
    Bucket=processed_bucket,
```

Продовження додатку А

```
Key=processed_key,
            Body=csv_buffer.getvalue()
        )
        print(f"Оброблені дані збережено як {processed_key} у бакеті {processed_bucket}")
```

Додаток Б

```
import numpy as np
import pandas as pd
import scipy
import sklearn
import joblib

print(f"NumPy версія: {np.__version__}")
print(f"Pandas версія: {pd.__version__}")
print(f"SciPy версія: {scipy.__version__}")
print("scikit-learn version:", sklearn.__version__)
print("joblib version:", joblib.__version__)

import boto3
import pandas as pd
from io import StringIO
s3 = boto3.client('s3')
bucket = 'air-pollution-processed-data'
# Отримуємо список об'єктів у бакеті
def list_files_in_s3(bucket, prefix=''):
    paginator = s3.get_paginator('list_objects_v2')
    pages = paginator.paginate(Bucket=bucket, Prefix=prefix)
    files = []
    for page in pages:
        if 'Contents' in page:
            for obj in page['Contents']:
                files.append(obj['Key'])
```

Продовження додатку Б

```
return files

files = list_files_in_s3(bucket)

# Завантажуємо всі CSV-файли та об’єднуємо їх у один DataFrame
dataframes = []
for file in files:
    if file.endswith(‘.csv’):
        response = s3.get_object(Bucket=bucket, Key=file)
        content = response[‘Body’].read().decode(‘utf-8’)
        df = pd.read_csv(StringIO(content))
        dataframes.append(df)

# Об’єднуємо всі дані
full_df = pd.concat(dataframes, ignore_index=True)

# Перевірка на пропущені значення
print(full_df.isnull().sum())

# Видалення рядків з пропущеними значеннями
clean_df = full_df.dropna()
# Перевірка на дублікатів
duplicates = clean_df.duplicated()
print(f"Кількість дублікатів: {duplicates.sum()}")
# Видалення дублікатів
```

Продовження додатку Б

```
clean_df = clean_df.drop_duplicates()

# Перетворення часових міток у зручний формат
clean_df['datetime'] = pd.to_datetime(clean_df['dt'], unit='s')

# Видалення стовпця 'dt' якщо він більше не потрібен
clean_df = clean_df.drop(columns=['dt'])

import matplotlib.pyplot as plt

# Встановимо індекс дати
clean_df.set_index('datetime', inplace=True)

# Введемо змінну для міста
city_name = 'Vinnytsia'

# Фільтруємо дані для вибраного міста
city_data = clean_df[clean_df['city'] == city_name]

# Побудова графіку
plt.figure(figsize=(12, 6))
plt.plot(city_data.index, city_data['aqi'], label='AQI')
plt.title(f'Рівень AQI у {city_name} з часом')
plt.xlabel('Дата')
plt.ylabel('AQI')
plt.legend()
plt.show()
```

Продовження додатку Б

```
import folium
from folium.plugins import HeatMap

# Створимо карту України
ukraine_map = folium.Map(location=[49.0, 32.0], zoom_start=6)

# Підготуємо дані для теплової карти
heat_data = [[row['lat'], row['lon'], row['aqi']] for index, row in
clean_df.iterrows()]
# Додамо теплову карту
HeatMap(heat_data).add_to(ukraine_map)
# Відобразимо карту
ukraine_map
# Виберемо лише числові колонки для аналізу кореляції
numeric_cols = clean_df.select_dtypes(include=['float64', 'int64']).columns
# Обчислимо кореляційну матрицю
corr_matrix = clean_df[numeric_cols].corr()

# Відобразимо теплову карту кореляцій
plt.figure(figsize=(12,10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Кореляційна матриця')
plt.show()
```

Продовження додатку Б

```
# Припустимо, що ми прогнозуємо AQI на основі історичних даних

# Виберемо дані для конкретного міста, наприклад, Київ
city_data = clean_df[clean_df['city'] == 'Vinnytsia'].copy()

# Сортуємо за датою
city_data.sort_index(inplace=True)

# Створимо лагові ознаки для часових рядів
for lag in range(1, 24):  # Лаги від 1 до 24 годин
    city_data[f'aqi_lag_{lag}'] = city_data['aqi'].shift(lag)
# Видалимо пропущені значення, що виникли через зсув
city_data = city_data.dropna()
# Розділимо на ознаки та цільову змінну
X = city_data[[f'aqi_lag_{lag}' for lag in range(1, 24)]]
y = city_data['aqi']
# Розділимо на тренувальну та тестову вибірки
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False,
test_size=0.2)
from sklearn.ensemble import RandomForestClassifier

# Створюємо модель
```

Продовження додатку Б

```
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Навчаємо модель
model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

y_pred = model.predict(X_test)
print("Точність моделі:", accuracy_score(y_test, y_pred))
print("Звіт про класифікацію:\n", classification_report(y_test, y_pred))

print('Матриця неточностей:')
print(confusion_matrix(y_test, y_pred))
# Візуалізація фактичних та прогнозованих значень
plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test, label='Фактичні')
plt.plot(y_test.index, y_pred, label='Прогнозовані')
plt.title('Порівняння фактичних та прогнозованих значень AQI')
plt.xlabel('Дата')
plt.ylabel('AQI')
plt.legend()
plt.show()
```

Продовження додатку Б

```
Точність моделі: 0.7241379310344828
Звіт про класифікацію:
              precision    recall  f1-score   support

         1.0       0.74      0.79      0.76        33
         2.0       0.70      0.64      0.77        25

    accuracy                           0.74        58
   macro avg       0.74      0.74      0.74        58
weighted avg       0.74      0.74      0.74        58
```

Рисунок 4.1.1 - Звіт про класифікацію

```
import joblib
import numpy as np

def model_fn(model_dir):
    model = joblib.load(os.path.join(model_dir, 'aqi_rf_model.joblib'))
    return model

def input_fn(request_body, request_content_type):
    if request_content_type == 'application/json':
        input_data = json.loads(request_body)
        features = np.array(input_data['features']).reshape(1, -1)
        return features
    else:
        raise ValueError("Тип контенту не підтримується")

def predict_fn(input_data, model):
    prediction = model.predict(input_data)
    return prediction
```

Продовження додатку Б

```
def output_fn(prediction, response_content_type):
    if response_content_type == ‘application/json’:
        result = {‘prediction’: int(prediction[0])}
        return json.dumps(result)
    else:
        raise ValueError("Тип контенту не підтримується")
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)