МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Аліна САВЧЕНКО
“__”_____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

((ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
“МАГІСТР”
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ “ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ

СИСТЕМИ ТА ТЕХНОЛОГІЇ”

Тема: “ Система прогностичного моделювання якості повітря на основі даних реальному часі ”

Виконавець: _____Євгеній Валерійович Василенко _____

Керівник: _____професор Зіатдінов Юрій Кашафович _____

Нормоконтролер: _____Ігор РАЙЧЕВ_____

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Комп’ютерних наук та технологій

Кафедра Комп’ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ

Завідувач випускової кафедри

Аліна САВЧЕНКО

«____ » _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Василенка Євгенія Валерійовича

1. **Тема роботи:** «Система прогностичного моделювання якості повітря на основі даних реальному часі» затверджена наказом ректора від 29 вересня 2023 за No 1976/ст.
2. **Термін виконання роботи:** 2 жовтня 2023 по 31 грудня 2023.
3. **Вихідні данні до роботи**: алгоритми машинного навчання; розробка веб-сторінок; завдання кластеризації та регресії даних.
4. **Зміст пояснювальної записки**: вступ, місце інтелектуального аналізу даних в машинному навчанні, алгоритми обробки даних, бібліотеки та інструменти для роботи з алгоритмами аналізу даних, створення веб сторінок.
5. **Перелік обов'язкового графічного матеріалу:** діаграми, слайди презентації.

6. **Календарний план-графік**

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	02.10.2023 – 10.10.2023	
2.	Огляд та аналіз наукової літератури по темі дипломної роботи та написання Розділу 1.	11.10.2023 – 27.10.2023	
3.	Написання Розділу 2 дипломної роботи.	28.10.2023 – 16.11.2023	
4.	Написання Розділу 3 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	17.11.2023 – 30.11.2023	
5.	Оформлення та друк пояснювальної записки.	01.12.2023 – 10.12.2023	
6.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	11.12.2023 – 12.12.2023	
7.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	13.12.2023 – 17.12.2023	

Дата видачі завдання: 02.10.2023 р.

Керівник дипломної роботи _____Юрій ЗІАТДІНОВ

(підпис керівника)

Завдання прийняв до виконання_____ Євгеній ВАСИЛЕНКО

(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Система прогностичного моделювання якості повітря на основі даних реальному часі»: 86 сторінок, 1 таблицю та 34 рисунка. Список бібліографічних посилань складається з 8 найменувань.

Ключові слова: ВИПАДКОВИЙ ЛІС, КЛАСТЕРИЗАЦІЯ, АЛГОРИТМ, РЕГРЕСІЯ, ВЕБ-СТОРІНКА, СЕРЕДОВИЩЕ РОЗРОБКИ.

Об’єкт дослідження: системи машинного навчання та аналізу даних, спрямовані на прогнозування якості повітря.

Предмет дослідження: алгоритм «випадкового лісу».

Мета роботи: дослідження роботи алгоритмів, які використовуються у машинному навчанні для різноманітних цілей. Як результат, на основі отриманих знань необхідно розробити систему для прогнозування якості повітря на основі даних отриманих у реальному часі. Додатковою метою є веб-додаток для відображення результатів роботи алгоритму.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- з’ясувати, наскільки тісно пов’язані алгоритми з машинним навчанням;
- розглянути найвідоміші алгоритми та їх логіку роботи;
- ознайомитись з методами інтеграції машинного навчання у свою систему;
- обрати платформу для розробки кінцевого продукту;
- розробити систему для прогнозування якості повітря на декілька днів вперед.

Методи дослідження: включають у себе:

- методи інтеграції алгоритмів;
- методи визначення точності роботи алгоритмів;
- методи порівняння кінцевого результату з актуальними даними.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

K-Means	–	тип кластеризації.
DBSCAN	–	алгоритм кластеризації.
EM	–	кластеризація очікувань-максимізації.
GMM	–	моделей суміші Гауса.
AdaBoost	–	метаалгоритм статистичної класифікації.
Випадковий ліс	–	алгоритм обробки великих даних та формування прогнозів.
AQI	–	індекс якості повітря.
ML	–	машинне навчання.
Python	–	сучасна мова програмування.
PRE	–	виконуваний файл Python, який інтерпретує код Python, пе-ретворюючи його на байт-код, який читається центральним процесором.
NumPy	–	це бібліотека Python, яка використовується для роботи з ма-сивами.
Ndarray	–	тип даних у бібліотеці NumPy.
SVM	–	мультиноміальна логістична регресія.
MVC	–	шаблон програмування у бібліотеці Django.
API	–	набір програмного коду, який забезпечує передачу даних між одним програмним продуктом та іншим.
CRUD	–	операції створення, читання, оновлення та видалення у базах даних.
Jinja	–	тип шаблону у Django.
POST	–	тип запиту, який використовується у роботі зв’язці сервер-клієнт.
SSL/HTTPS	–	протоколи захисту.
Series	–	тип даних у бібліотеці Pandas.

PyCharm IDE	–	середовище розробки програмного коду.
K-Means	–	тип кластеризації.
DBSCAN	–	алгоритм кластеризації.
EM	–	кластеризація очікувань-максимізації.
GMM	–	моделей суміші Гауса.
AdaBoost	–	метаалгоритм статистичної класифікації.
Випадковий ліс	–	алгоритм обробки великих даних та формування прогнозів.
AQI	–	індекс якості повітря.
ML	–	машинне навчання.
Python	–	сучасна мова програмування.
PRE	–	виконуваний файл Python, який інтерпретує код Python, пе-ретворюючи його на байт-код, який читається центральним процесором.
NumPy	–	це бібліотека Python, яка використовується для роботи з масивами.
Ndarray	–	тип даних у бібліотеці NumPy.
SVM	–	мультиноміальна логістична регресія.
MVC	–	шаблон програмування у бібліотеці Django.
API	–	набір програмного коду, який забезпечує передачу даних між одним програмним продуктом та іншим.
CRUD	–	операції створення, читання, оновлення та видалення у базах даних.
Jinja	–	тип шаблону у Django.
POST	–	тип запиту, який використовується у роботі зв’язці сервер-клієнт.
SSL/HTTPS	–	протоколи захисту.
Series	–	тип даних у бібліотеці Pandas.
PyCharm IDE	–	середовище розробки програмного коду.

ВСТУП

Зараз перед людством гостро постає питання забруднення навколишнього середовища. Робляться спроби збереження чистоти та очищення води, землі , повітря. Забруднення повітря є великою проблемою серед забруднень навколишнього середовища, якщо не найбільшою. Усьому живому організму на землі потрібно повітря, чисте повітря. Жодна з живих істот не може вижити без чистого повітря. Але людська діяльність, у вигляді сільськогосподарської діяльності, автомобілів, роботи промисловість, видобуток корисних копалин, спалювання палива та сміття забруднює повітря. Завдяки цій діяльності в нашому повітрі поширюються різні діоксиди сірки, чадний газ, тверді частинки пилу та інше, що є шкідливим для всього живого організму

Забруднене повітря, яким ми дихаємо, викликає багато проблем зі здоров'ям як людей так і тварин, зменшує імунітет. Це призводить до потреби прогнозування цих забруднень щоб допомогти покращити навколишнє середовище. Це примушує нас шукати нові методи прогнозування забруднення повітря. Отже, ми для нашого розумного міста вирішуємо задачу прогнозування забруднення повітря за допомогою різних технології аналізу даних.

Найбіше для цього підходить техніка багатовимірного багатокрокового аналізу даних часових рядів із використанням методів інтелектуальних аналізу даних. Наша система буде використовувати данні що змінюються у часі(минулі та поточні) та застосовувати їх для прогнозування забруднення повітря. Ця буде зменшувати складність і підвищувати ефективність і практичність, а також забезпечить більш надійні та точні висновки (рішення) для організацій охорони навколишнього середовища та розумного міста

РОЗДІЛ 1
МЕТОДИ АНАЛІТИЧНОГО ПРОГНОЗУВАННЯ ЗАБРУДНЕННЯ ПОВІТРЯ

1.1. Огляд проблеми інтелектуального аналізу даних

В сучасному інформаційному світі види і об`еми різних даних, які формуються та зберігаються щодня, зростають у вигляді експоненти. Недавнє дослідження підрахувало, що щохвилини Google отримує понад 2 мільйони запитів (і це число постійно збільшується), користувачі електронної пошти надсилають понад 200 мільйонів повідомлень, користувачі YouTube завантажують 48 годин відео, користувачі Facebook діляться понад 680 000 частинами вмісту, а користувачі Twitter створюють 100 000 твітів. Також різні ресурчи обміну медіа даннними, сайти для торгівлі та джерела новин постійно генерують та накопичують нові дані протягом дня. Декілька років тому, коли людство почало використовувати великі об`еми інформації, «великі дані», щоб знаходити закономірності в них та нові ідеї, майже одразу з'явилася нова взаємопов'язана область досліджень неструктурованих великих об`емів данних: інтелектуальний аналіз даних.

1.2. Інтелектуальний аналіз даних та його задачі

Інтелектуальний аналіз даних означає вилучення або видобуток знань із великих обсягів даних [1]. Іншими словами, інтелектуальний аналіз даних — це наука, мистецтво та технологія виявлення великих і складних масивів даних з метою винаходження корисних шаблонів. Теоретики та практики постійно шукають удосконалені методи, щоб зробити процес більш ефективним, економічно ефективним і точним.

Багато інших термінів мають подібне або дещо інше значення для інтелектуального аналізу даних, наприклад, видобуток знань із даних, вилучення знань, аналіз даних/шаблонів, вичерпування даних.

Інтелектуальний аналіз даних розглядається як синонім іншого широко використовуваного терміна, виявлення знань з даних або ВЗД. Інші розглядають інтелектуальний аналіз даних просто як важливий крок у процесі відкриття знань, у якому застосовуються інтелектуальні методи для вилучення шаблонів даних [1].

Грегорі П’ятецький-Шапіро ввів термін «виявлення знань у базах даних» у 1989 році. Однак термін «інтелектуальний аналіз даних» став більш популярним у бізнесі та пресі. Зараз інтелектуальний аналіз даних і виявлення знань використовуються як взаємозамінні.

Зараз інтелектуальний аналіз даних використовується практично в усіх місцях, де зберігається та обробляється велика кількість даних [2].

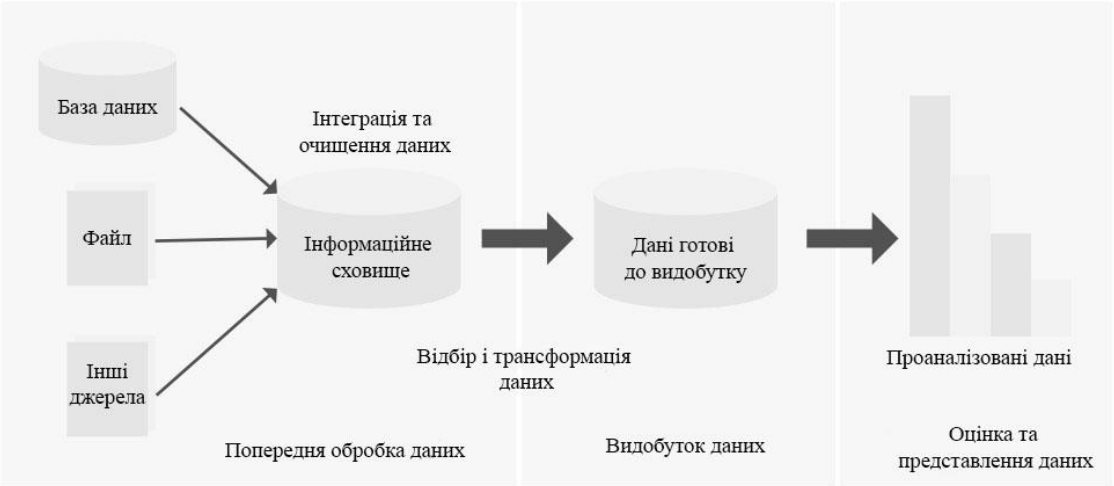

Рис. 1.1. Процес обробки даних за допомогою інтелектуального аналізу

Виявлення нових закономірностей із даних складається з таких кроків:

- очищення даних [2];
- інтеграція даних [2];
- вибір даних [2];
- трансформація даних [2];
- інтелектуальний аналіз даних [2];
- оцінка зразків [2];
- презентація знань (візуалізація данних для представлення отриманих знань користувачеві).

1.3. Підхід «контрольоване навчання»

Доволі поширеною проблемою класифікації є контрольоване навчання, тому що мета часто полягає в тому, щоб змусити комп’ютер вивчити систему класифікації, яку ми обрали чи створили. Типовим прикладом класифікаційного навчання є розпізнавання цифр. Вивчення класифікації підходить для вирішення будь-якої проблеми, де виведення класифікації має користь і легко визначити класифікацію. Інколи,в деяких випадках, при навчанні може не знадобитися давати

кожному випадку проблеми попередньо визначені класифікації, якщо агент може розробити класифікації для себе. Це, приклад навчання, так званий, без контролю в контексті класифікації.

Контрольоване навчання може залишати частину вхідних даних невизначеною. Якщо деякі з вхідних значень відсутні, неможливо зробити обґрунтовані висновоки про вихідні дані, алє ця модель не потрібна, поки доступні вхідні дані. При неконтрольованому навчанні припускється, що всі спостереження спричинені латентними змінними, іншими словами це значить що вхідні данні знаходяться в наприкінці причинно-наслідкового ланцюга.

Найпоширенішим методом навчання нейронних мереж і дерев рішень є контрольоване навчання. Ці методи, обидва, сильно залежать від данних, які надходять з попередньо визначені класифікації. У випадку нейронних мереж класифікація використовується для визначення помилки мережі в процесі навчання, а потім налаштування мережі для ii зменьшення, а в деревах рішень класифікації використовуються для визначення того, які атрибути та надають більше важливої інформації, яку можна використовувати для вирішення класифікаційної задачі. В процесі будуть розглянуті обидва підходи детальніше, але наразі буде достатньо знати, що обидва ці приклади розвиваються завдяки контролю у формі заздалегідь визначених класифікацій. Індуктивне машинне навчання — це процес вивчення набору правил із екземплярів (прикладів у навчальному наборі), або, загалом кажучи, створення класифікатора, який можна використовувати для узагальнення з нових екземплярів. Послідовність застосування керованого машинного навчання, відповідно,до проблеми в реальному світі описано на рисунку 1.2. Першим кроком є збір початкового набору даних. При наявності експерта, то він може підказати, які поля (атрибути, ознаки) є найбільш значущими, інформативними.

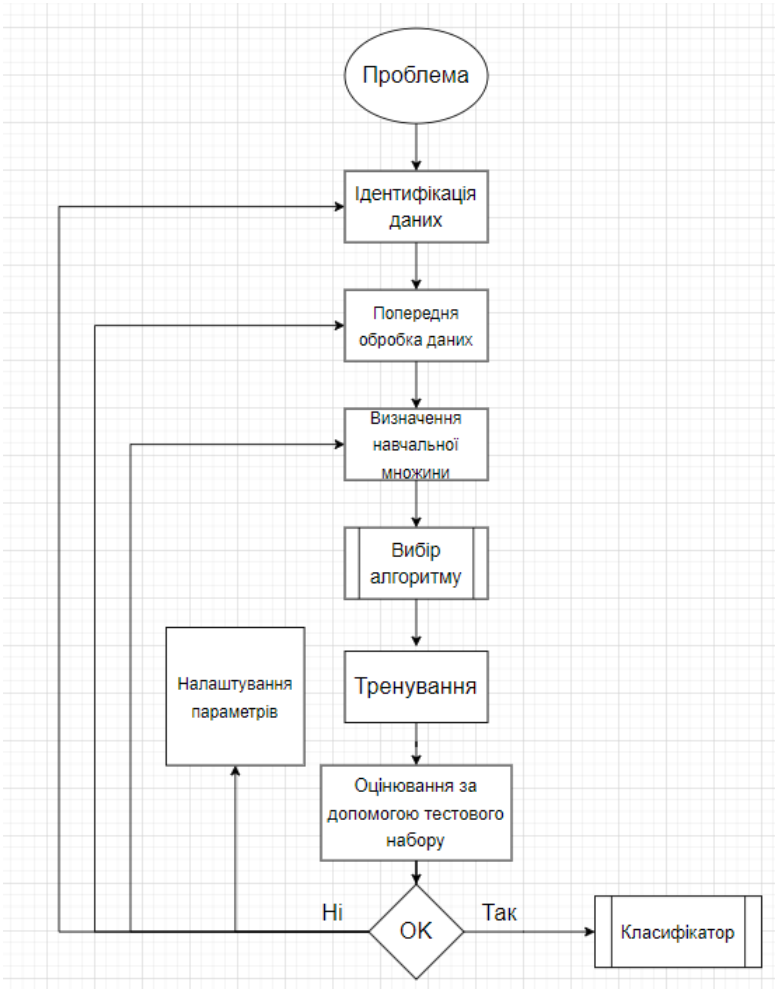

Рис. 1.2. Процес застосування керованого машинного навчання

1.4. Підхід «неконтрольованого навчання»

Навчання без нагляду виглядає складнішим: мета це навчити комп’ютер робити те, чого ми не говоримо йому робити. Насправді при неконтрольованному навчанні існує два підходи:

- максимізація нагороди - полягає в тому, щоб навчати комп’ютер не за допомогою чіткої класифікації, а за допомогою певної системи винагороди для визначення успішного рішення.

Необхідно зауважити, що при цьому навчанняі, як правило, прийняття рішення вписується в структуру проблеми, оскільки метою є не створення класифікації, а прийняття рішень, які збільшують винагороду.

Часто форма навчання з підкріпленням може бути використана для навчання без нагляду, коли користувач базує свої дії на попередніх винагородах і покараннях, навіть не обов’язково дізнаючись будь-яку інформацію про те, як його дії впливають на світ. У певному сенсі вся попередня інформація непотрібна, так як користувач вже знає про винагороду і що робити, оскільки, вивчає залежність винагороди яку він очікує отримати за кожну дію, тобто вже знає, що робити без будь-якої обробки яку він повинен був виконати. Це має велику вагу у ситуаціях, коли обчислення кожної можливості займає дуже багато часу (навіть якщо є відомості всі ймовірності переходу між станами середовища). Але також зрозуміло, що іншого боку, навчання методом проб і помилок, з обробкою всіх можливостей може зайняти дуже багато часу та ресурсів. Навчання з обробкою всіх можливостей може бути більш ефективним, оскільки не має передумови попередньої класифікації прикладів. У певних випадках наші класифікації, серед вже відомих можуть бути не найкращими. Одним із яскравих прикладів є те, що загальноприйняте уявлення про гру в шахи перевернулося, коли серія комп’ютерних програм (спеціалізованих комп’ютерів), які пройшли навчання просто граючи одна проти іншої, або сама з собою, знову і знову, шляхом неконтрольованого навчання, стала сильнішою за найкращих шахістів,. Ці програми виявили деякі принципи, які призвели до здивування експертів з гри в шахи, і працювали краще, ніж програми гри в шахи, навчені на попередніх прикладах.

- Кластеризація - це другий тип неконтрольованого навчання, у якому метою навчання є, а просто пошук схожості в прикладах навчальних даних а не максимізація функції корисності.

Часто роблять такі припущення, що виявлені кластери досить добре збігаються з очевидною, зрозумілою, класифікацією. Для прикладу, якщо кластеризувати людей використовуючи демографічні данні, це призведе, наприклад, до кластеризації заможних в одній групі та бідних в іншій. Незважаючи на те, що наш алгоритм не знае, не має опису цих кластерів, він може створювати їх, а в подальшому використовувати ці описи кластерів для створення та призначення одному чи іншому з кластерів нових прикладів. При цьому підході, керованному

даними, який зазвичай здатний працювати добре, коли є достатньо даних для обробки (наприклад обробка соціальної інформації, такі як ті, що Amazon.com використовує для рекомендації книг,в пошукових системах відбір відповідей то що, засновані на принципі пошуку схожих вподобань груп людей і подальшого використання (заповнення новими користувачами) групам. У певних ситуаціях, наприклад, із обробкою тафільтрацією соціальної інформації, данних про інших членів кластера (наприклад, які книги вони читають) може вистачати для того, щоб ми отримали з алгориту значущі результати. Хоча зазвичай ці нові кластери є просто корисними для людини-аналітика, є інструментом загальногоаналізу. Цяж проблема присутня і при неконтрольованому навчанні, проблема переобладнання навчальних даних. Покищо не винайшли ніякого алгоритму, щоб уникнути проблеми, оскільки будь-який алгоритм, який може навчатися на своїх вхідних даних, має бути досить потужним.

1.5. Метод аналізу даних «кластеризація»

Кластеризуючий аналізом – це один із методів інтелектуального аналізу даних, аба ще називають «чисельною таксономією.» Цей метод по суті групує великі обсяги даних разом на основі їх подібності [5]. На рис. 1.3 показано можливий вигляд аналізу кластеризації.

Рис. 1.3. Кластеризуючий аналіз

Дані, які спорадично відображаються на діаграмі, можна згрупувати стратегічно за допомогою аналізу кластеризації. Цей аналіз зазвичай використовується як попередня обробка (або може діяти як попередня обробка), при я кому данні форматуються таким чином, щоб можна було до них застосувати інші методи.

Існує п’ять основних методів, підходів до кластеризації які використовуються зараз при дослідженні даних:

- алгоритми поділу: створення різних поділів і подальше їх оцінювання на основі конкретних критеріїв [5];
- алгоритми ієрархії: створення ієрархічного розташування набору даних за допомогою певних критеріїв [5];
- на основі щільності: на основі функцій зв’язності та щільності [3];
- на основі сітки: на основі багаторівневих структур деталізації [3];
- на основі моделі: спочатку створюється гіпотеза моделі для кожного з кластерів, потім визначається найкраща відповідність моделі [3];

Власне ці алгоритми кластеризації використовуються для визначення класу кожної точки початкових даних у певну групу. Визначаються властивості або

характеристики для оремих груп, і точки даних групуються по мають схожим властивостям. Зараз це алгоритми такі:

- кластеризація K-Means: групує спостереження в кластери, за принципом коли де точки даних додаються до кластера з найближчим середнім значенням;
- кластеризація за середнім зсувом: кластерам іте-раційно шляхом зміщення призначає точки даних у бік режиму. Найчастіше використовується в розпізнанні та обробці зображень;
- просторова кластеризація додатків із шумом на основі щільності (DBSCAN): об’єднує точки даних у певну группу, якщо вони близько розташовані одна до одної, одночасно позначаючи у областях з низькою щільністю в кластері конкретні точки викидів;
- кластеризація очікувань-максимізації (EM) за допомогою моделей суміші Гауса (GMM): враховує дисперсію (ширина дзвоноподібної кривої) для кластеризації немаркованих даних, для визначення форми розподілу або кластера;
- агломеративна ієрархічна кластеризація: робить аналіз «знизу вгору» що створює, за підходом, ієрархічний аналізу кластерів. Спостереження ведуться в окремих кластерах, а пари кластерів об’єднуються в міру просування по ієрархії «знизу вгору».

Існує багато способів при кластерному аналізі отримати дані. Наприклад, в страхових компаніях можуть визначити страхувальників із високим середнім розміром вимог і формувати з нихгрупи. В маркетингу для сегментації клієнтів кластеризацію можна використовувати для обрання продукту на основі переваг, які вони отримають, купуючи його. Іншим прикладом кластеризації є розробка маршрутів евакуації при землетрусах, сейсмологи можуть аналізувати землетруси і силу кожного землетрусу, а потім використовують ці дання при побудові маршрутів.

1.6. Метод класифікації даних

Класифікація — це обробка пошуку набору моделей (або функцій), які описують і розрізняють класи даних або концепції, з метою використання моделі для

прогнозування класу об’єктів, мітка класу яких невідома. Визначена модель залежить від дослідження набору інформації про навчальні дані (тобто об’єктів даних, мітка класу яких відома). Похідна модель може бути представлена в різних формах, таких як правила класифікації (якщо – тоді), дерева рішень і нейронні мережі. Інтелектуальний аналіз даних має багато типів класифікатора, для прикладу розглянемо «Дерево рішень» та «Байєсовську класифікацію» [4].

Дерево рішень — це структура дерева, подібна до блок-схеми, де кожен вузол представляє перевірку значення атрибута, кожна гілка позначає результат перевірки, а листя дерева представляють класи або розподіл класів [4]. Правила класифікації легко трансформуються з дерев рішень. Цєй підхід не вимагає жодних попередніх припущень щодо типу розподілу ймовірностей, іншими словами, якому задовольняють клас та інші атрибути. Тобто залучення дерева рішень є непараметричною методологією яка використовується при побудові різних моделей класифікації. Дерева рішень легко інтерпретувати, особливо якщо вони відносно малого розміру. З точки зору точності класифікації дерева можна порівнянна з двома іншими методами для простих наборів даних. Для вивчення дискретизованих функцій вони забезпечують добре представлення, хоча є типи данних які вони погано спрощують, або певні типи булевих задач.

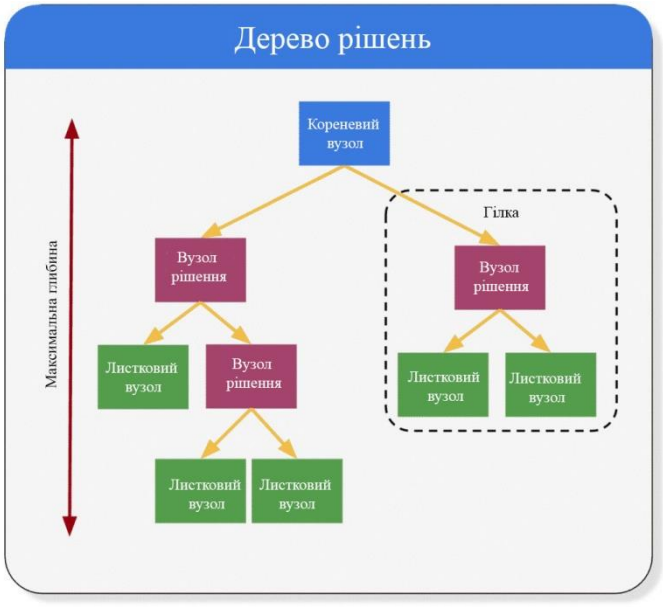

Рис. 1.4. Принцип побудови дерева рішень

Байєсівські класифікатори — статистичні класифікатори [6]. Вони працюють з ймовірностямі належності, тобто роблять припущення про приналежность до класу, наприклад, ймовірність того, на основі теореми Байєса робиться припущення про те що дана вибірка належить до заданого класу. Байєсовська класифікація створена. Порівнюючи алгоритми класифікації, в процесі досліджень, виявилось, що простий байєсівський класифікатор, також який називають «наївний байєсівський» класифікатор, за продуктивністю знаходится на рівні з деревом рішень і також класифікаторами нейронної мережі. При використанні байєсовські класифікаторів при роботі з великими базами даних можна отримати високу швидкість та точність.

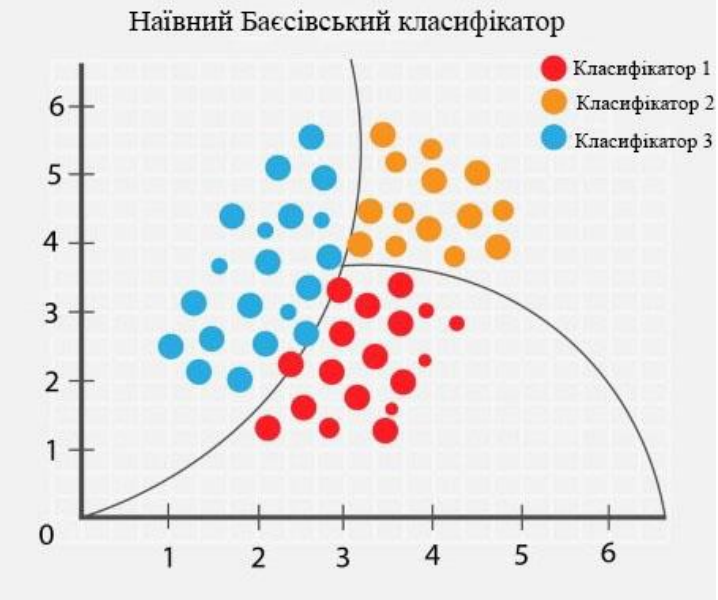

Рис. 1.5. Наївний Баєсівський класифікатор

Наївні байєсівські класифікатори приймають, що точне значення атрибута в даному класі не залежить від значень інших атрибутів. Це припущення називається умовною незалежністю класу [6]. Це зроблено для спрощення розрахунків і вважається «наївним». Мережі байєсівських вірувань — це графічні копії, які, на відміну від наївних байєсівських класифікаторів, дозволяють відображати залежності між підмножинами атрибутів.

1.7. Метод виявлення викидів

Метод виявлення викидів, також відомий як виявлення аномалій, цей метод інтелектуального аналізу даних діє протилежно кластеризації. Замість пошуку великих груп даних, які можна об'єднати разом, виявлення викидів шукає точки даних, які є рідкісними та виходять за межі встановленої групи чи середнього значення [8].

Рис.1.6. Метод виявлення аномалій

Оскільки дані доволі випадкові, аномалії не обов'язково вказують на тенденцію. Натомість дані, які суперечать дійсності, можуть вказувати на те, що відбувається щось ненормальне та потребує подальшого аналізу. Коли компанія чи організація виявляють ці аномалії в даних, стає легше зрозуміти, чому виникають ці аномалії, і підготуватися до будь-яких, що можуть виникнути в майбутньому [8].

Існує два типи викидів (аномалій):

- однофакторний: точка даних, яка складається з крайнього значення однієї змінної [7];

- багатоваріантність: комбінація незвичайних балів принаймні за двома змінними [7];

При аналізі випадкових данних, знаходження або виявлення аномалій (викидів) також активно використовується, наприклад при виявленні шахрайства. Типовий ариклад із банківської сфери це аналіз активності використання клієнтом банківської картки, виявлення підозрілої активності, нетипових дій може дати банку іфновмацю про необхідність деяких дій (блокування рахунку, то що.). Цей же підхід використовується при виявленні вторгнення (кібербезпека). Аналіз викидів дозволяє передчасно помітити спроби порушення даних на веб-сайтах, щоб їх можна було оперативно усунути. Зараз це дуже актуальне питання, у зв'язку поширеням кібер атак.

1.8. Підвищення і AdaBoost

Підвищення — це техніка ансамблевого моделювання, яка намагається побудувати сильний класифікатор із числа слабких класифікаторів. Це робиться з Використовуються шлях послідовності слабких класифікаторів у побудови моделі. Першим кроком будується модель з навчальних даних. Другим кроком будується друга модель, в якій ми намагаємося виправити виявлені помилки першої моделі. Ця послідовність повторюється, додаються нові моделі, до тих пір, коли не буде достатньо точно передбачено увесь набір навчальних даних або дійдемо до обмеження максимального числа моделей. AdaBoost був першим дійсно успішним алгоритмом підвищення, розробленим для бінарної класифікації. AdaBoost є скороченням від Adaptive Boosting і є дуже популярною технікою підвищення, яка поєднує кілька «слабких класифікаторів» в один «сильний класифікатор». Його сформулювали Йоав Фройнд і Роберт Шапір. Вони також отримали премію Геделя 2003 року за свою роботу.

Послідовніс дій в алгоритмі:

1. створення набору даних з однаковою вагою кожній точці даних;

2. призначити данні з п.1 як вхідні для наступної моделі та пошук неправильно класифікованих точок даних;
3. збільшення вагу некоректних точок даних;
4. якщо модель достатньо точна, то перейти до п. 5, інакше - до п. 2;
5. завершення.

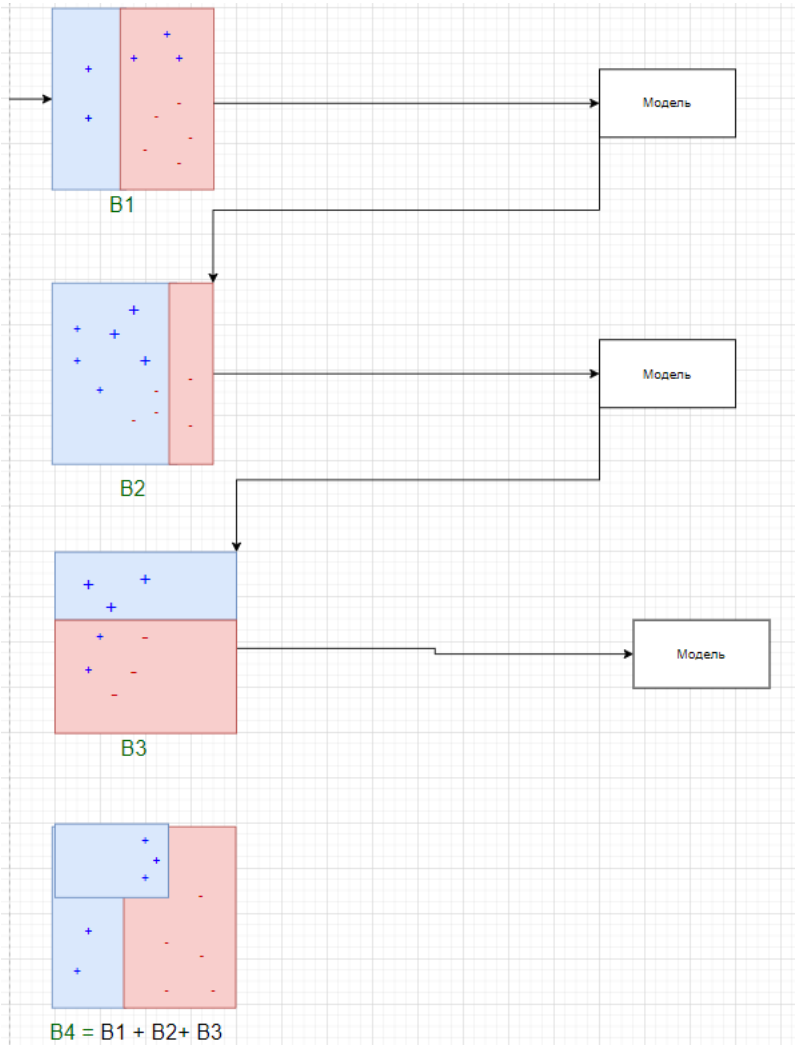

Рис. 1.7. Принцип роботи алгоритму «AdaBoost»

На рисунку 1.7. дуже показано алгоритм AdaBoost. Розглянемо цей алгоритм детальніше в поетапному процесі:

- B1 складається з 10 точок даних, які складаються з двох типів, а саме плюс(+) і мінус(-), 5 з яких є плюс(+), а інші 5 – мінус(-), і кожній з них спо-чатку було присвоєно однакову вагу. Перша модель намагається класифікувати точки

даних і генерує вертикальну роздільну лінію, але вона неправильно класифікує 3 плюс (+) як мінус (-);

- B2 складається з 10 точок даних із попередньої моделі, у яких 3 неправильно класифіковані плюси (+) мають більшу вагу, щоб поточна модель більше намагалася класифікувати ці плюси (+) правильно. Ця модель генерує вертикальну роздільну лінію, яка правильно класифікує раніше неправильно класифіковані плюси (+), але в цій спробі вона неправильно класифікує три мінуси (-);
- B3 складається з 10 точок даних із попередньої моделі, у яких 3 неправильно класифіковані мінуси (-) мають більшу вагу, щоб поточна модель більше намагалася правильно класифікувати ці мінуси (-). Ця модель генерує горизонтальну роздільну лінію, яка правильно класифікує раніше неправильно класифіковані мінуси (-);
- B4 об'єднує B1, B2 і B3, щоб побудувати сильну модель прогнозування, яка є набагато кращою, ніж будь-яка окрема використана модель.

1.9. Висновки до розділу 1

Машинне навчання викоритовує контрольований або неконтрольованим підхід. При малій кількості даних і чітко позначені дані для навчання краще себе показує контрольоване навчання. Неконтролбоване навчання проявляє себе краще при роботі з великими наборами даних. Також виявилось що, краще використовувати методи глибокого при наявності величезних наборів даних. В розділі були розглянуті питання навчання з підкріпленням, глибоке навчання з підкріпленням. У цьому розділі були роглянуті деякі види алгоритмів машинного навчання. На теперішній час машинне навчання використовуеться в багатьох сферах непомітно для людини. Від результатів пошуку рекомендованих для до, наприклад, сортування та вибір медіа у соціальних мережах. У цьому розділі є необхідний базис до розуміння більшості популярних алгоритмів навчання. Наприклад, є статистичні моделі для оцінки, алє вони непрацюють добре коли є відсутні значення та велики точки даних.

Враховуючи вище сказане, ми бачимо важливість машинного навчання у сучасному світі, його застосування у багатьох сферах людського життя. Ми отримуємо рішення задач аналізу данних, розпізнавання об’єктів, обробка природної мови, прогнозування подій на основі попередніх данних та інше. Тому вибір підходу у начанні стає важливим питанням, так як більшість алгоритмів показують добрі результати, вони точно ідентифікують атрибут, але їх ефетиініть різна на різних типах данних.

РОЗДІЛ 2
ВИБІР ТА ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ ДЛЯ ЇЇ РЕАЛІЗАЦІЇ

2.1. Вибір алгоритму для створення системи, його можливості

2.1.1. Алгоритм «Випадковий ліс»

Випадковий ліс — це популярний алгоритм машинного навчання, який належить до методики навчання під наглядом [3]. Його можна використовувати як для задач класифікації, так і для регресії в машинному навчанні. Він базується на концепції ансамблевого навчання, яке є процесом поєднання кількох класифікаторів для вирішення складної проблеми та покращення продуктивності моделі. Як випливає з назви, «Випадковий ліс» — це класифікатор, який містить кілька дерев рішень на різних підмножинах заданого набору даних і бере середнє значення для підвищення точності прогнозування цього набору даних». Замість того, щоб покладатися на одне дерево рішень, випадковий ліс бере прогнози з кожного дерева на основі більшості голосів прогнозів і прогнозує кінцевий результат [3].

Більша кількість дерев у лісі підвищує точність і запобігає проблемі переобладнання. Рисунок 2.1. пояснює роботу алгоритму випадкового лісу.

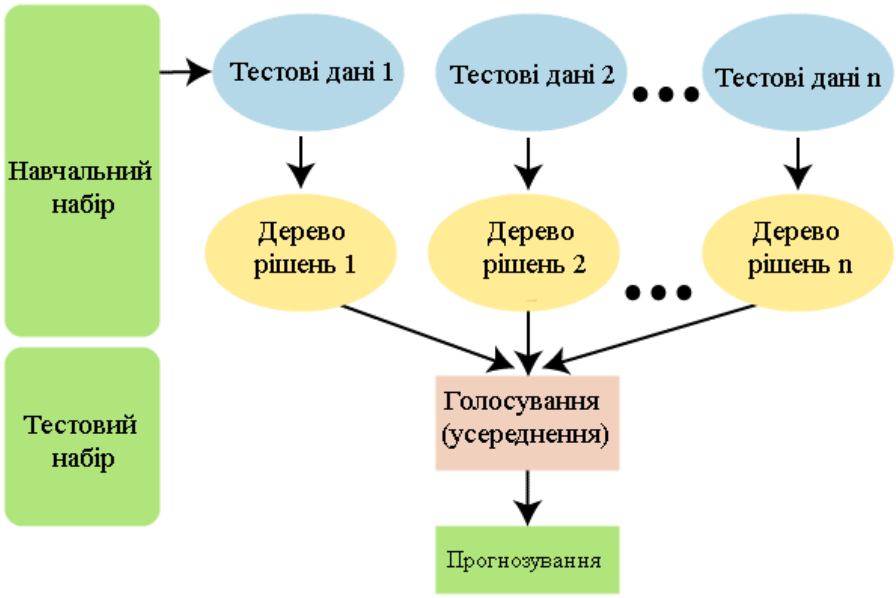

Рис. 2.1. Схема роботи алгоритму «Випадковий ліс»

Оскільки випадковий ліс об'єднує кілька дерев для прогнозування класу набору даних, можливо, що деякі дерева рішень можуть передбачити правильний вихід, а інші – ні. Але разом усі дерева передбачають правильний результат. Тому нижче наведено два припущення для кращого класифікатора випадкових лісів:

- у змінній функції набору даних повинні бути деякі фактичні значення, щоб класифікатор міг передбачити точні результати, а не припущення [3];
- прогнози з кожного дерева повинні мати дуже низькі кореляції [3].

Випадковий ліс працює у два етапи: перший полягає у створенні випадкового лісу шляхом об'єднання N дерев рішень, а другий – у створенні прогнозів для кожного дерева, створеного на першому етапі. Робочий процес можна пояснити за допомогою наступних кроків і схеми:

- крок 1: обрати випадкові K точок даних із навчального набору;
- крок 2: побудувати дерева рішень, пов'язані з вибраними точками даних (підмножини);
- крок 3: вибрати число N для дерев рішень, які потрібно побудувати;
- крок 4: повторити кроки 1 і 2;

- крок 5: для нових точок даних знайти прогнози для кожного дерева рішень і призначити нові точки даних категорії, яка виграє більшість голосів.

Роботу алгоритму можна краще зрозуміти на наступному прикладі: припустімо, що є набір даних, який містить кілька зображень фруктів. Отже, цей набір даних передається класифікатору «випадкового лісу». Набір даних розділено на підмножини та надано кожному дереву рішень. Під час фази навчання кожне дерево рішень дає прогнозований результат, і коли з'являється нова точка даних, на основі більшості результатів класифікатор «Випадковий ліс» прогнозує остаточне рішення.

2.1.2. Індекс якості повітря

Якість повітря означає стан повітря навколо нас. Хороша якість повітря означає, наскільки повітря чисте, чисте та вільне від забруднюючих речовин, таких як дим і пил, а також інших газоподібних домішок у повітрі. У таблиці 2.1 наведено основні типи забруднюючих речовин разом із коротким описом кожного. Хороша якість повітря є необхідною умовою для збереження крихкої рівноваги життя на землі для людей, рослин, тварин і природних ресурсів, і вони знаходяться під загрозою, коли забруднення в повітрі досягає критичних концентрацій.

Таблиця 1.1

Основні види забруднюючих речовин.

CO	Оксид вуглецю в основному від спалювання природного газу, вугілля або деревини
CO2	Вуглекислий газ природний і необхідний на стабільному рівні
SOx	Оксиди сірки від вулканів і промисловості є однією з причин для занепокоєння щодо екологічного впливу використання цих видів палива як джерел енергії
NOx	Оксиди азоту є побічним продуктом двигунів внутрішнього згоряння, які використовуються в транспорті та промисловості
PM	PM2,5 означає, що діаметр твердих часток менше 2,5 мікрон, а для PM10 – 10 мікрон. Це дуже залежить від місцевих умов, таких як клімат, дорожній рух і забруднення.
O3	Озон — це парниковий газ, який утворюється в результаті реакції сонячного світла на повітря. Вуглеводні та оксиди азоту в повітрі реагують з утворенням озону безпосередньо біля джерела забруднення або за багато кілометрів за вітром
WOC	Летючі органічні сполуки, такі як метан, надзвичайно ефективні парниковий газ, який сприяє посиленню глобального потепління

2.1.3. Часовий ряд

Часовий ряд – це, по суті, послідовність точок даних, виміряних протягом певного часу. Вимірювання в часовому ряді розташовані в хронологічному порядку та складаються або з однієї змінної, яка називається однофакторною, або з кількох залежних змінних, які називаються багатофакторною. Безперервні часові ряди – це спостереження, виміряні в усіх випадках часу, тоді як дискретні включають вимірювання в окремі моменти часу. Зазвичай дискретний набір складається з послідовних спостережень, записаних через рівні проміжки часу, як-от години, щоденні чи річні розриви. Вважається, що на часовий ряд впливають чотири основні компоненти: тенденція, сезонність, циклічність і нерегулярність. Тренд - це загальна тенденція змін у часовому ряді з довгостроковим рухом зростання, зменшення або стагнації. Сезонні коливання пов’язані зі змінами протягом сезонів року, де клімат і погода є важливими факторами. Циклічна частина описує відмінності як середньострокові зміни в часовому ряді, спричинені обставинами циклічного характеру. Останнім компонентом, нерегулярним, є випадкові варіації, або так звані шуми, які не є типовими і не можуть бути описані попередніми частинами. З цими чотирма компонентами існує два різних типи моделей, які використовуються для часових рядів: мультиплікативний і адитивний.

Припущення для моделі мультиплікативного часового ряду полягає в тому, що всі компоненти не обов’язково є незалежними і можуть впливати один на одного, якщо адитив припускає незалежність компонентів.

Стаціонарність — ще одна концепція часових рядів. Часовий ряд є стаціонарним, якщо його властивості, такі як середнє значення та дисперсія, не залежать від часу. Таким чином, стаціонарність є корисним припущенням через меншу математичну складність моделі. Часовий ряд може бути гомоскедастичним, що стосується зразків часового ряду, які мають дисперсію подібності, як і будь-які інші зразки в наборі даних. Гетероскедастичне означає протилежне з різною дисперсією протягом усього часового ряду. Підсумовуючи: стаціонарні та гомоскедастичні часові ряди ведуть себе добре, і їх легше передбачити, а

нестаціонарні та гетероскедастичні ряди набагато складніші. В останньому випадку, застосовуючи математичні перетворення, щоб зробити ряд стаціонарним і гомоскедастичним, щоб зменшити точну проблему.

2.1.4. Машинне навчання

У своєму дослідженні я буду використовувати машинне навчання. Машинне навчання – це галузь, яка походить від штучного інтелекту. Його мета полягає в тому, щоб дозволити комп'ютеру навчатися самостійно, без явного програмування правил. Алгоритм машинного навчання може ідентифікувати та вивчати основні закономірності в спостережених даних, щоб моделювати та прогнозувати світ. Існує три типи техніки машинного навчання: навчання з підкріпленням, навчання без нагляду та навчання під наглядом. У навчанні з підкріпленням алгоритм отримує зворотний зв'язок на основі ефективності, коли він переміщується у своєму проблемному просторі. Такі завдання, як гра в гру або водіння автомобіля, є прикладами, коли навчання з підкріпленням підходить. Неконтрольоване навчання – це підхід, який вивчає дані, які не мають позначок або секретні. Замість того, щоб реагувати на зворотний зв'язок, як у навчанні з підкріпленням, неконтрольоване навчання визначає спільні атрибути та характеристики з даних. Алгоритми неконтрольованого навчання включають проблеми асоціації, які намагаються описати частини даних, і проблеми кластеризації, які прагнуть визначити природні групи. У контрольованому навчанні алгоритм намагається навчатися на інформативних прикладах позначених даних. Такі алгоритми можна описати як підхід, керований даними, де історичні дані використовуються для прогнозів майбутнього. Прогнозування якості повітря часто вирішується за допомогою контрольованих методів, оскільки часові ряди можуть перетворюватися на позначені пари вхідних і вихідних даних, де цільовим вихідним сигналом є основна правда наступного значення в послідовності даних.

2.1.5. Розділення набору даних

Перед навчанням моделі машинного навчання набір даних зазвичай розбивається на набір даних для навчання, перевірки та тестування. Навчальний набір даних призначений для навчання моделі та складатиметься з більш суттєвої частини набору даних. Після того, як модель навчилася з навчального набору даних, модель працює з окремим набором даних перевірки. Цей набір перевірки є меншою підмножиною навчальних даних і може використовуватися для оцінки під час навчання. Нарешті, тестовий набір даних містить невидимі дані для навченої моделі. Навчальний набір даних має оцінити, чи модель узагальнила набори даних замість запам’ятовування результату.

Різні стратегії, пов’язані з тим, як найефективніше розділити набір даних, є важливими для досягнення кращих узагальнених результатів. Перехресна перевірка — це техніка, яка включає кілька раундів поділу наборів даних на підмножини. Раунди, так звані K-згортки, виконуються кілька разів, щоб зменшити мінливість продуктивності моделі та дати кращу оцінку прогнозної ефективності моделі. Хоча ця процедура потребує багато часу та обчислень із збільшенням складок, модель перевірила кожну точку даних, що зменшує зміщення, а також дисперсію оцінюють кілька разів. Мета полягає в тому, щоб узагальнити вимірювання ефективності від навчання на наборі тестів до прогнозів на основі невидимих даних.

2.2. Вибір мови програмування та бібліотек. Загальний огляд, причини вибору для реалізації завдання

2.2.1. Мова програмування Python

Якщо ви багато працюєте за комп’ютером, з часом ви виявите, що є завдання, які ви хочете автоматизувати. Наприклад, ви можете виконати пошук і заміну у великій кількості текстових файлів або перейменувати та змінити порядок файлів фотографій у складний спосіб. Можливо, ви захочете написати невелику спеціальну

базу даних, або спеціалізований додаток з графічним інтерфейсом користувача, або просту гру.

Python простий у використанні, але це справжня мова програмування, яка пропонує набагато більше структури та підтримки для великих програм, ніж сценарії оболонки або пакетні файли. З іншого боку, Python також пропонує набагато більше перевірки помилок, ніж C, і, будучи мовою дуже високого рівня, він має вбудовані типи даних високого рівня, такі як гнучкі масиви та словники. Завдяки більш загальним типам даних Python застосовний до значно більшої проблемної області, ніж Awk або навіть Perl, але багато речей у Python принаймні так само прості, як і в цих мовах.

Python дозволяє вам розділити нашу програму на модулі, які можна повторно використовувати в інших програмах Python. Він постачається з великою колекцією стандартних модулів, які ви можете використовувати як основу своїх програм — або як приклади, щоб почати вчитися програмувати на Python. Деякі з цих модулів надають такі речі, як файловий ввід/вивід, системні виклики, сокети та навіть інтерфейси для наборів інструментів графічного інтерфейсу користувача.

Python — це інтерпретована мова, яка може заощадити вам значний час під час розробки програми, оскільки не потрібні компіляція та зв’язування. Інтерпретатор можна використовувати в інтерактивному режимі, що дозволяє легко експериментувати з особливостями мови, писати одноразові програми або тестувати функції під час розробки програми знизу вгору.

Python дозволяє писати програми компактно та зрозуміло. Програми, написані на Python, зазвичай набагато коротші, ніж еквівалентні програми на C, C++ або Java, з кількох причин:

- типи даних високого рівня дозволяють виражати складні операції в одному операторі;
- групування тверджень здійснюється за допомогою відступів замість початкових і кінцевих дужок;
- не потрібні оголошення змінних або аргументів.

Python є розширюваним: якщо ви знаєте, як програмувати на C, ви можете легко додати нову вбудовану функцію чи модуль до інтерпретатора, або для виконання критичних операцій з максимальною швидкістю, або для зв'язування програм Python із бібліотеками, які можуть бути лише доступними у двійковій формі (наприклад, графічна бібліотека певного виробника).

2.2.2. Контроль пам'яті у Python

Система пам'яті є основним фактором, що визначає продуктивність [2]. Процес розподілу, звільнення та ефективного керування пам'яттю називається керуванням пам'яттю. В основному це стосується виділення пам'яті, коли це потрібно програмі, а потім звільнення її, коли вона більше не потрібна. Оскільки неможливо оцінити, скільки пам'яті потрібно для запуску програми, потрібен додатковий код для керування мінливими вимогами до пам'яті. Існує два пов'язані завдання керування пам'яттю:

- розподіл: виділення пам'яті за запитом програми для блоку пам'яті. Цей блок пам'яті отримується від операційної системи. Частина менеджера пам'яті, яка виконує це, відома як розподільник;
- переробка: після того, як блоки пам'яті були виділені, але більше не потрібні, вони переробляються або за допомогою ручного керування пам'яттю, у якому це вирішує програміст, або автоматичного керування пам'яттю, у якому цю роботу виконує менеджер пам'яті [2].

Python — мова програмування високого рівня, реалізована мовою C. Він обробляє все у формі об'єктів, наприклад, список, словник, кортеж, цілі числа зберігаються як об'єкт. Усі ці структури даних і об'єкти Python зберігаються в приватній купі, якою керує менеджер пам'яті Python. Чотири поняття, пов'язані з менеджером пам'яті Python:

- купа: це сукупність усієї керованої пам'яті Python;
- арени: це найбільші блоки пам'яті, кожен з яких має фіксований розмір 256 КБ. Python запитує їх у системи, і вони є об'єктами, які складають купу;

- пули: це блоки пам’яті, які складають арени по 4 КБ кожна. Вони є просто масивами через їхні фіксовані розміри.
- блоки: об’єкти Python зберігаються в блоках. Вони мають певний формат на основі типів даних. Наприклад, ціле число займає більше місця, ніж символ, для ефективності використовується інший розмір блоку.

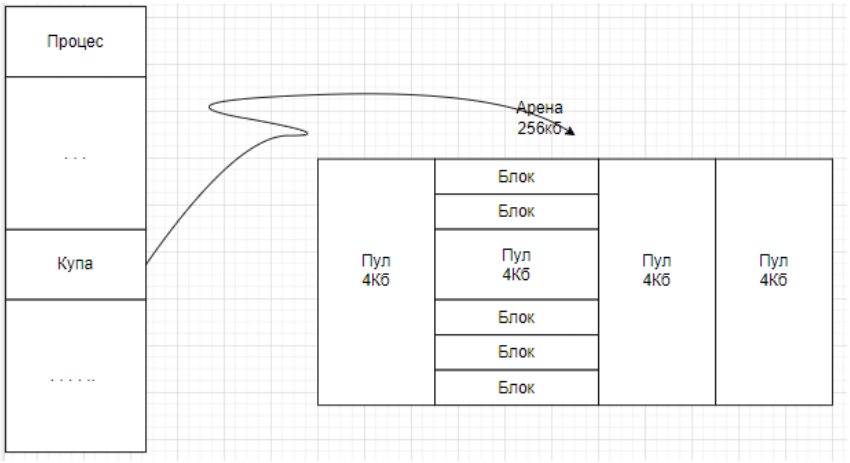

Рис. 2.2. Зв’язок між купою, аренами, пулами та блоками

Пам'ять може бути звільнена для ОС лише у формі арени, тому, коли арена порожня, вона звільняється. З тієї ж причини, щоразу, коли Python запитує пам’ять, арена, яка вже заповнена, буде надана, якщо можливо, замість порожньої. Однак щоразу, коли блок звільняється, він не буде випущено в ОС, а збережеться для подальшого розподілу пам’яті. Це звільнення пам'яті виконується автоматично мовою Python через збирач сміття [2]. Звільнення пам’яті за допомогою збирача сміття є процесом, протилежним ручному управлінню пам’яттю, у якому програміст має вказати, які об’єкти звільнити [2]. За замовчуванням порогові значення для збирача сміття Python встановлені на 700, 10, 10 для трьох поколінь зібраних об’єктів. Він автоматично збиратиметься, коли буде досягнуто порогових значень. Він також відстежує всі виділення та звільнення пам’яті, і якщо кількість виділень мінус кількість звільнень досягає 700, об’єкт або видаляється, якщо на нього більше немає посилань, або його переміщують до наступного покоління, якщо він все ще має посилання. Те ж саме повторюється для поколінь 2 і 3, але з нижніми порогами

10 [3]. Цей процес далі поширюється на нижній рівень, де інтерпретатор (CPython, PyPy тощо) взаємодіє з диспетчером пам’яті ОС, щоб мати справу з пам'яттю.

Коротше кажучи, весь процес можна сформулювати таким чином, коли виконується будь-яка програма, написана на Python, вона компілюється до проміжного коду, що називається байт-кодом, який потім інтерпретується Python Runtime Environment (PRE) у машинний код. Крім того, виділення та звільнення пам’яті обробляється збирачем сміття, керованим PRE. Щоразу, коли створюється новий об’єкт, необхідна пам’ять виділяється, і як тільки об’єкт виходить за межі своєї області, пам’ять стає придатною для збирання сміття та зрештою буде звільнена цією службою [3].

2.2.3. Бібліотека NumPy. Призначення та основні функції

NumPy — це бібліотека Python з відкритим кодом, яка використовується майже в усіх галузях науки та техніки. Це універсальний стандарт для роботи з числовими даними в Python, і він лежить в основі наукових екосистем Python і PyData. Серед користувачів NumPy — усі, від початківців програмістів до досвідчених дослідників, які проводять найсучасніші наукові та промислові дослідження та розробки. API NumPy широко використовується в Pandas, SciPy, Matplotlib, scikit-learn, scikit-image та більшості інших наукових і наукових пакетів Python.

Бібліотека NumPy містить багатовимірні масиви та матричні структури даних. Він надає ndarray, однорідний n-вимірний об’єкт масиву, з методами для ефективної роботи з ним. NumPy можна використовувати для виконання різноманітних математичних операцій над масивами. Він додає до Python потужні структури даних, які гарантують ефективні обчислення з масивами та матрицями, і надає величезну бібліотеку математичних функцій високого рівня, які працюють з цими масивами та матрицями.

NumPy надає два фундаментальних об’єкти: об’єкт N-вимірного масиву (ndarray) і об’єкт універсальної функції (ufunc). Крім того, існують інші об’єкти, які

створюються на основі цих, які можуть бути корисними для нашої роботи, і про них буде сказано пізніше.

N-вимірний масив — це однорідна колекція «елементів», проіндексованих за допомогою N цілих чисел. Є дві важливі частини інформації, які визначають N-вимірний масив: форма масиву та вид елемента, з якого складається масив. Форма масиву — це кортеж з N цілих чисел (по одному для кожного виміру), який надає інформацію про те, наскільки індекс може змінюватися в цьому вимірі. Інша важлива інформація, яка описує масив, — це тип елемента, з якого складається масив. Оскільки кожен N-вимірний масив є однорідною колекцією точно однакових типів даних, кожен елемент займає блок пам’яті однакового розміру, і кожен блок пам’яті в масиві інтерпретується точно так само.

У NumPy ndarray — це N-вимірний масив елементів, де кожен елемент займає фіксовану кількість байтів. Як правило, ця фіксована кількість байтів представляє число (наприклад, ціле чи з плаваючою комою). Однак ця фіксована кількість байтів також може представляти довільний запис, що складається з будь-якої колекції інших типів даних. NumPy досягає такої гнучкості завдяки використанню об’єкта типу даних (dtype). З кожним масивом пов’язаний об’єкт dtype, який описує макет даних масиву. Кожен об’єкт dtype, у свою чергу, має пов’язаний об’єкт типу Python, який точно визначає, який тип об’єкта Python повертається під час доступу до елемента масиву. Об’єкти dtype є достатньо гнучкими, щоб містити посилання на масиви інших об’єктів dtype, і тому їх можна використовувати для визначення вкладених записів.

Кожен об’єкт dtype базується на одному з 21 вбудованого об’єкта dtype. Ці вбудовані об’єкти дозволяють виконувати числові операції з широкою різноманітністю цілих чисел, даних із плаваючою комою та складних типів даних. З кожним типом даних пов’язаний об’єкт типу Python, екземпляри якого є скалярними масивами. Цей тип-об’єкт можна отримати за допомогою атрибута type об’єкта dtype. Python зазвичай визначає лише один тип даних певного класу даних (один тип цілого числа, один тип із плаваючою комою тощо). Це може бути зручно для деяких програм, яким не потрібно турбуватися про всі способи представлення даних у

комп’ютері. Однак для наукових застосувань це не завжди вірно. Як наслідок, у NumPy вбудовано 21 різний фундаментальний об’єкт дескриптора типу даних Python. Ці дескриптори здебільшого базуються на типах, доступних у мові C, на якій написаний CPython. Однак є кілька типів, які є надзвичайно гнучкими, наприклад str, unicode та void.

2.2.4. Переваги та недостатки використання бібліотеки NumPy

Звичайно, як і кожного існуючого рішення для певних проблем існують його переваги та недостатки. Нижче наведено пункти, які пояснюють переваги NumPy:

- Ядром Numpy є його масиви. Однією з головних переваг використання масивів Numpy є те, що вони займають менше пам’яті та забезпечують кращу швидкість виконання порівняно з подібними структурами даних у Python (списки та кортежі).
- Numpy підтримує деякі спеціальні наукові функції, такі як лінійна алгебра. Вони допомагають розв'язувати лінійні рівняння.
- Numpy підтримує векторизовані операції, такі як поелементне додавання та множення. Списки Python не підтримують ці функції.
- Це дуже хороша заміна MATLAB, OCTAVE тощо, оскільки вона забезпечує подібні функції та підтримку з швидшою розробкою та меншими розумовими витратами (оскільки Python легко писати та розуміти)
- NumPy дуже хороший для аналізу даних.

До недостатків NumPy можна віднести наступні пункти:

- Використання «nan» у Numpy: «Nan» означає «не число». Він був розроблений для вирішення проблеми відсутніх значень. Сам NumPy підтримує «nan», але відсутність міжплатформної підтримки в Python ускладнює роботу користувача. Ось чому ми можемо зіткнутися з проблемами під час порівняння значень в інтерпретаторі Python.

- Вимагає безперервного розподілу пам’яті: операції вставлення та видалення стають дорогими, оскільки дані зберігаються в безперервних розташуваннях пам’яті, оскільки їх зміщення вимагає зміщення.

2.2.5. Бібліотека Scikit-learn. Короткий огляд та призначення

Scikit-learn — це ключова бібліотека для мови програмування Python, яка зазвичай використовується в проектах машинного навчання. Scikit-learn зосереджено на інструментах машинного навчання, включаючи математичні, статистичні та алгоритми загального призначення, які є основою для багатьох технологій машинного навчання. Як безкоштовний інструмент Scikit-learn надзвичайно важливий у багатьох різних типах розробки алгоритмів для машинного навчання та пов’язаних технологій.

Деякі з основних ключових елементів Scikit-learn, корисних для машинного навчання, включають алгоритми класифікації, регресії та кластеризації. Наприклад, Scikit-learn підтримує роботу з випадковими лісами, де окремі цифрові дерева містять інформацію про вузли, яка об’єднується в декілька деревних архітектур для досягнення лісового підходу. Інший спосіб поговорити про це полягає в тому, що кожне дерево включає кластеризовані вузли в топології дерева, а аналіз різних дерев додається разом, щоб отримати глобальний підхід, який точніше обробляє дані для показу результатів.

Окрім випадкового лісу, Scikit-learn допомагає з посиленням градієнта, векторними машинами та іншими елементами машинного навчання, які є ключовими для досягнення результатів. Як основний ресурс Scikit-learn працює з такими інструментами, як SciPy і matplotlib, які забезпечують візуалізацію та багато іншого.

2.2.6. Перетворення даних в Scikit-learn

Трансформація даних є вирішальним кроком у будь-якому аналізі даних. Для різних вхідних змінних часто використовуються загальні перетворення (наприклад, перетворення відбілювання, диференціація та логарифмічне співвідношення шансів). Scikit-learn надає кілька зручних функцій для виконання цих методів перетворення та попередньої обробки даних. Основною структурою даних, яка використовується майже у всіх функціях Scikit-learn, є масив NumPy. Незалежні змінні, також відомі як функції в спільноті машинного навчання, представлені масивом $N_o \times N_f$ з N_o як кількістю спостережень і N_f як кількістю функцій. Залежні змінні, які також називають цілями або мітками, представлені масивом $N_o \times N_l$, де N_l представляє кількість міток. У більшості програм для навчання під наглядом ми маємо справу з однією міткою з декількома значеннями, тому N_l дорівнює одиниці. За домовленістю, велика літера X використовується для позначення масиву ознак, а нижня літера y використовується для позначення масиву міток.

2.2.7. Контрольоване навчання в Scikit-learn

Контрольоване навчання відноситься до підмножини алгоритмів машинного навчання, які встановлюють відображення між змінними функції та їхніми відповідними цільовими змінними. Передумовою використання методів навчання під наглядом є відомість як функцій, так і відповідних позначок. Контрольоване навчання можна розділити на дві категорії на основі характеру міток, регресію для безперервних міток і класифікацію для дискретних міток. У цьому розділі я зосереджусь на випадку класифікації, і узагальнення до регресії є формально тривіальним шляхом простої заміни викликів функцій класифікації на виклики функцій регресії. Щоб зробити вступ зрозумілим, я зосереджуюсь на чотирьох популярних класифікаторах, а саме на SVM, максимальної ентропії (також називається мультиноміальною логістичною регресією), штучній нейронній мережі та випадковому лісі.

Природне питання, яке виникає, полягає в тому, який із цих методів слід використовувати на практиці. Це просте запитання, але, на жаль, однозначної відповіді немає. Каруана та Нікулеску-Мізіл (2006) [3] провели обширне емпіричне дослідження, щоб порівняти низку методів навчання під керівництвом на основі емпіричних даних, і їхній висновок полягав у тому, що ефективність методів значною мірою залежить від конкретного набору даних, хоча SVM і випадковий ліс є найефективнішими для ряду завдань класифікації.

Робочий процес, реалізований у Scikit-learn для використання класифікатора, включає три кроки. Спочатку створюється модель, де вказуються гіперпараметри моделі. В другому кроці, зіставляється модель з навчальними даними та вивчаються параметри. І в кінці кінців, застосовується підібрана модель до тестових даних, щоб отримати прогнозовані мітки. На рисунку 2.3 зображений код-приклад для цього робочого процесу.

```
model = classifier(hyper-parameters = something)
model.fit(X_train, y_train)
y_test = model.predict(X_test)
```

Рис. 2.3. Приклад використання класифікатора

2.3. Бібліотека Django. Опис основних можливостей бібліотеки.

Django було представлено в 2003 році, коли Адріан Головатий і Саймон Віллісон почали використовувати Python для створення програм [4]. Мотивацією для розробки Django була необхідність виконувати програми за розкладом днів або навіть годин. Через два роки після початку створення фреймворку веб-додатків, у 2005 році, він був випущений як відкритий код. Сьогодні у Django кілька десятків тисяч користувачів і показники розвитку. Оскільки Django є фреймворком веб-додатків на основі Python, для його використання потрібен Python. Python постачається з легким сервером для тестування, але для належного використання майбутньому сайту на основі Django потрібен сервер Apache. Django офіційно підтримує бази даних PostgreSQL, MySQL, Oracle і SQLite. [4]

Django також вільно застосовує архітектуру MVC (model-view-controller). По суті, models.py, views.py і urls.py, а також шаблон HTML template.Models.py містять опис інформації та надають функції для створення, отримання, оновлення та видалення даних. View.py передає HTML-шаблон вмісту сайту. Роль файлу urls.py, у свою чергу, полягає в тому, щоб налаштувати, який файл представлення викликається. Завдяки такій структурі та функціям файлів для редагування програми не потрібно редагувати багато файлів, що, у свою чергу, економить час.

Django, про який часто говорять як про базову структуру веб-додатків, дійсно має багато функцій і підтримки. Набагато меншу програму можна завантажувати в основну програму, щоб вони спілкувалися один з одним за допомогою простих API. Він має свої правила і спосіб реалізації функціоналу. Як показано на рисунку 2.4, організація різних розділів програм упакована в окремий каталог. У цих каталогах ми можемо побачити «admin.py», «forms.py», «models.py» і «views.py», «urls.py». Це стандартні назви файлів, з яких Django намагається знайти ресурси. Більшість реалізації, пов’язаної з базою даних, знаходиться на models.py і views.py, що складається з функцій обслуговування кінцевих точок і urls.py, що містить маршрути.

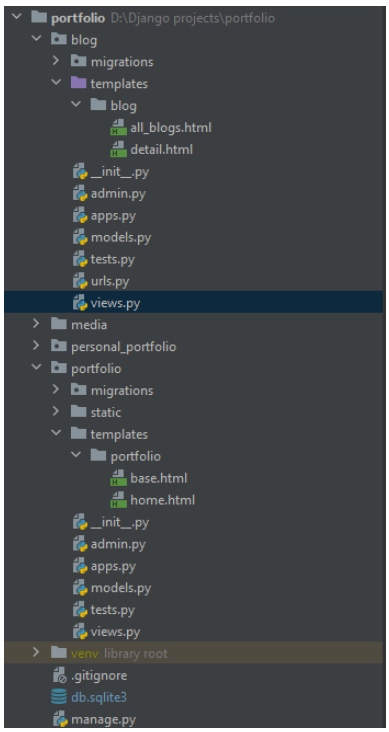

Рис. 2.4. Структура проекту в Django

2.3.1. Моделі в Django

Типовий спосіб для програм Django взаємодіяти з даними — через моделі Django. Модель Django — це об’єктно-орієнтований клас Python, який представляє характеристики сутності. Наприклад, сутність може бути особою, компанією, продуктом або іншою концепцією, яку використовує програма. Оскільки дані є центром сучасних додатків, а фреймворк Django підтримує принцип DRY (не повторюй себе), моделі Django часто служать будівельними блоками для проектів Django. Якщо є набір моделей Django, що представляють сутності програми, моделі Django також можуть служити основою для спрощення створення інших конструкцій Django, які працюють з даними (наприклад, формами, представленнями на основі класів, службами REST, сторінками адміністратора Django), звідси випливає важливість моделей Django в цілому.

Основна технологія зберігання Django є реляційною (тобто вона може підключатися до SQLite, Postgres, MySQL або Oracle), тому модель Django розроблена для прямого зіставлення з таблицею реляційної бази даних. Це означає, що екземпляри моделі Django зберігаються як рядки реляційної таблиці, названої на честь моделі.

Наприклад, для класу моделі Django під назвою Store за замовчуванням Django виконує операції бази даних CRUD (створення, читання, оновлення та видалення) у таблиці бази даних під назвою <назва_додатку>_store, де кожен із екземплярів моделі Store представляє рядки бази даних і поля моделі (наприклад, ім’я, адреса, місто, штат) зіставляються зі стовпцями таблиці бази даних. Оскільки моделі Django обертаються навколо даних, вони схильні до змін. Наприклад, модель Django під назвою Store може раптово вимагати модифікації вихідних полів через бізнес-вимоги (наприклад, додавання нового поля, наприклад електронної пошти, або видалення поля, яке більше не потрібно, як телефон). Збереження цих змін у моделях Django протягом тривалого часу також є важливим аспектом моделей Django, яким керують за допомогою файлів міграції використання.

2.3.2. Django шаблони. Їх синтакс

Шаблони Django визначають макет і остаточне форматування, які надсилаються кінцевим користувачам після завершення обробки запиту методом перегляду.

Хоча існує понад 100 вбудованих конструкцій, які допомагають вам створювати шаблони Django – для початку, це найважливіші елементи синтаксису, які потрібно знати:

- {{вихідна_змінна}} – значення, оточені подвійними фігурними дужками на початку та в кінці, представляють вихідні змінні. Змінні передаються представленнями Django, параметрами посилання або контекстними процесорами в шаблони. У шаблоні можна використовувати «{{}}» для виведення вмісту змінної, а також використовувати крапкову нотацію Python для виведення глибших елементів змінної (наприклад, полів, ключів, методів). Наприклад, {{магазин.ім’я }} повідомляє шаблону Django вивести назву змінної «магазин», де «магазин» може бути об’єктом і називати поле, або «магазин» може бути словником і називати ключ.
- {% тег %} – значення, укладені у фігурні дужки зі знаками відсотка, називаються тегами. Теги Django пропонують складну логіку форматування, укладену в просте синтаксичне представлення.
- змінна|фільтр – значення, оголошені після вертикальної риски | називаються фільтрами. Фільтри Django пропонують спосіб застосування логіки форматування до окремих змінних. Будь-який інший синтаксис у шаблонах Django, окрім цих трьох варіантів, розглядається «як є». Це означає, що якщо в шаблоні оголошено заголовок мови гіпертекстової розмітки (HTML) <h1>Ласкаво просимо!</h1>, користувач отримає великий заголовок HTML. Це дуже просто.

2.3.3. Шаблони Jinja в Django

Окрім шаблонів Django, платформа Django також підтримує шаблони Jinja. Jinja — це окремий проект механізму шаблонів, який дуже схожий на вбудовану систему шаблонів Django. Однак впровадження та зростання шаблонів Jinja в проектах Django частково пояснюється обмеженнями дизайну шаблонів Django, які майже не змінилися з моменту створення Django.

Для того, щоб отримати загальне уявлення про шаблони Jinja та дізнатися, чи добре вони підходять для проектів Django, ось деякі основні переваги та недоліки шаблонів Jinja. Почнемо з переваг:

- швидкість і продуктивність. Jinja компілює вихідний код шаблону в байт-код Python під час першого завантаження, тому шаблон аналізується лише один раз, що забезпечує кращу продуктивність під час виконання. Крім того, Jinja також підтримує опцію попередньої компіляції, що також може призвести до кращої продуктивності. Хоча швидкість і продуктивність є одними з найбільш дискусійних переваг шаблонів Jinja, враховуючи багато факторів, що впливають на швидкість і продуктивність (наприклад, запити до бази даних/навантаження, конфігурація сервера). Загалом, за рівних умов, шаблон Jinja, який виконує те саме, що й шаблон Django, версія Jinja буде швидшою за версію Django;
- гнучкість. Шаблони Jinja дуже гнучкі щодо того, що вони можуть містити, підтримуючи такі концепції, як макроси та інші конструкції, схожі на Python. Хоча деякі з цих практик не рекомендуються у веб-шаблонах, можна оцінити у використанні деякі з цих функцій, які недоступні або суворо обмежені в шаблонах Django;
- подібність до шаблонів Django. Jinja фактично натхненний шаблонами Django, тому між двома системами є багато спільного. Потужні функції, як-от успадкування шаблонів і блоки, працюють однаково, тому для використання Jinja в проектах Django потрібно менше часу, ніж можна собі уявити. Крім

того, функції безпеки (наприклад, автоматичне екранування) також тісно інтегровані в Jinja, як і в шаблонах Django;

- асинхронне виконання. Шаблони іноді можуть завантажувати багато даних або використовувати функції, які потребують багато часу для виконання, що спричиняє затримки в шаблонах, які повинні «чекати» завершення резервних завдань (тобто, вони синхронні). Шаблони Jinja підтримують асинхронне виконання, що дозволяє завданням підтримки виконувати свій курс – без шаблонів утримування – і пізніше знову збиратися з шаблонами після завершення. Однак необхідно зауважити, що ця функція вимагає використання асинхронних генераторів, які доступні лише в Python 3.6 або новіших версіях.

А тепер деякі недоліки шаблонів Jinja:

- підтримка сторонніх пакунків практично відсутня. Оскільки офіційна підтримка Django для шаблонів Jinja з'явилася відносно недавно – починаючи з Django 1.8, попередньої версії з довгостроковою підтримкою (LTS) до Django 1.11, майже вся сторонні пакунки (наприклад, Django admin) все ще створюються за шаблонами Django. Це може ускладнити створення проекту Django із чистим шаблоном Jinja та вимагати, щоб шаблони Jinja співіснували разом із шаблонами Django, що, у свою чергу, може призвести до труднощів і плутанини, коли потрібна настройка шаблону;
- нові поняття. Якщо ви звикли до шаблонів Django, деякі функції Jinja вимагають додаткової практики для розуміння та правильного використання (наприклад, макроси Jinja, фільтри Jinja). Хоча це не повинно бути проблемою, якщо ви новачок у Django загалом, оскільки кожна концепція нова і потребує певної практики.

2.3.4. Веб-запити та маршрутизація в Django

Django обробляє запити своєрідно. У Django об'єкт запитів має бути явно наданий функції перегляду або класу. Тоді об'єкт запиту містить усю необхідну

інформацію від поточного стану програми до поточного сеансу. Коли клієнт запитує ресурс із програми, створюється об'єкт HttpRequest і викликається відповідна функція перегляду. Якщо об'єкт запиту потрібно змінити або отримати доступ, його потрібно явно надати реалізованій функції. Як показано на рисунку 2.5., контекст запиту явно передається через функцію «post» представлення на основі класу або доступ до нього через область класу.

```
class VariationListView(LoginRequiredMixin, ListView):
    model = Variation
    queryset = Variation.objects.all()

    def post(self, request):
        formset = VariationInventoryFormSet(request.POST, request.Files)
        if formset.is_valid():...
        raise Http404
```

Рис. 2.5. Приклад реалізації POST запиту в Django

У Django маршрутизація обробляється через диспетчер URL-адрес. У файлі views.py записується функція/клас перегляду та URL-адреса з необхідним шаблоном. Диспетчер URL-адрес за замовчуванням уже можна налаштувати в налаштуваннях. Після налаштування кореневої URL-адреси можна додати інші маршрути URL-адрес. Ці маршрути можуть бути глобальними або специфічними для програми. Маршрути можна налаштувати в «urls.py», як показано на рисунку 2.6. «VariationListView» викликається, коли користувач робить будь-який запит на кінцеву точку, яка описана наступним чином: http://0.0.0.0:5000/products/1/inventory.

```
urlpatterns = [
    url(r'^(P<pk>\d+)/inventory/$' VariationListView.as_view, name='product_inventory')
]
```

Рис. 2.6. Приклад налаштування маршрутизації

2.3.5. Безпека в Django

Безпека є найважливішим аспектом будь-якої веб-програми. Вона відіграє важливу роль в управлінні та збереженні інформації кінцевого користувача. Існують різні типи проблем веб-безпеки та загроз, які ховаються в Інтернеті. Щодня можна знайти нові загрози та вразливості. Не всі ризики пом’якшуються негайно, і не відомі майбутні загрози. Однак запровадження профілактичних заходів і використання найкращих практик — це те, за що повинна гарантуватись структура веб-додатків. Це пов’язано з тим, що користувачі використовують веб-програму, а конфіденційна інформація користувача має бути недоторканою на сервері. Оскільки Django є проектом з відкритим кодом, усі аспекти безпеки фреймворку продумані. Він охоплює такі питання безпеки:

- захист від підробки міжсайтового запиту (CSRF);
- міжсайтовий сценарій (XSS);
- SQL ін'єкція;
- клікджекінг;
- перевірка заголовка хосту;
- SSL/HTTPS.

Система шаблонів Django захищає програму від більшості атак XSS. У випадку CSRF він має проміжне програмне забезпечення CSRF, яке перевіряє, чи походить заголовок запиту з того самого джерела. Django має вбудовану ORM, яка допомагає запитувати базу даних без жорсткого кодування коду SQL. Ці запити та набори запитів створюються за допомогою параметризації запиту. За допомогою цих стандартних наборів запитів більшість проблем із впровадженням SQL уже вирішено. З точки зору проблеми клікджекінгу, фреймворк постачається з проміжним програмним забезпеченням під назвою X-Frame-Options. У Django є можливість налаштувати SSL/HTTPS, щоб наскрізні запити та відповіді були зашифровані.

2.3.6. База даних SQLite

SQLite — це вбудована бібліотека, яка реалізує самодостатню систему транзакційної бази даних SQL, яка не потребує конфігурації. Вихідний код для SQLite існує у відкритому доступі та є безкоштовним як для приватних, так і для комерційних цілей. SQLite має прив'язки до кількох мов програмування, таких як C, C++, BASIC, C#, Python, Java та Delphi. Обгортка COM (ActiveX), яка робить SQLite більш доступним для скриптових мов у Windows, таких як VBScript і JavaScript, таким чином додаючи можливості до програм HTML. Вона також доступна у вбудованих операційних системах, таких як iOS, Android, Symbian OS, Maemo, Blackberry та WebOS, завдяки своєму малому розміру та простоті використання. SQLite — це вбудована бібліотека, яка реалізує механізм бази даних SQL. Код для SQLite є суспільним надбанням, тому його можна вільно використовувати для будь-яких цілей, комерційних чи приватних. Наразі SQLite можна знайти в більшій кількості додатків, включаючи кілька відомих проектів. SQLite — це вбудована система баз даних SQL, яка не має окремого серверного процесу, як більшість інших баз даних SQL. SQLite читає та записує безпосередньо у звичайні дискові файли. Формат файлу бази даних є кросплатформним. Ці функції роблять SQLite популярним вибором як формат файлу програми. SQLite — це компактна бібліотека, розмір бібліотеки може бути менше 500 КБ, залежно від цільової платформи та налаштувань оптимізації компілятора. Якщо додаткові функції опущено, розмір бібліотеки SQLite можна зменшити до 300 КБ. SQLite також можна змусити працювати в мінімальному стеку (4 КБ). SQLite — популярний вибір механізму баз даних для гаджетів з обмеженою пам'яттю, таких як мобільні телефони, КПК та MP3-плеєри. SQLite зазвичай працює швидше. Тим не менш, продуктивність зазвичай досить хороша навіть у середовищах з низьким обсягом пам'яті. SQLite дуже ретельно тестується і перед кожним випуском також є дуже надійним. Більша частина вихідного коду SQLite присвячена виключно тестуванню та перевірці. SQLite витончено реагує на збої розподілу пам'яті та помилки дискового введення/виведення. Звичайно, навіть з усім цим тестуванням все ще є помилки. Але

SQLite відкрита і чесна щодо всіх помилок і надає списки помилок, включаючи списки критичних помилок і щохвилинну хронологію звітів про помилки та змін коду.

Інфраструктура SQLite складається з чотирьох основних частин: ядро, компілятор SQL, бекенд (серверна частина), аксесуари:

- Основна частина містить інтерфейс користувача, командний процесор SQL і віртуальну машину. Інтерфейс користувача складається з бібліотеки функцій і структур C для виконання таких операцій, як ініціалізація баз даних, виконання запитів і перегляд результатів. Виклики функцій, які виконують SQL-запити, використовують командний процесор SQL. Командний процесор працює точно так само, як компілятор. Під час виконання програми віртуальна машина направляє потік керування через великий оператор switch, який переходить до блоку коду на основі поточного коду операції.
- Компілятор SQL містить токенізатор, парсер і генератор коду. Коли має бути виконано рядок, що містить інструкції SQL, інтерфейс передає цей рядок до токенізера. Завдання токенізера полягає в тому, щоб розбити вихідний рядок на токени та передавати ці токени один за одним аналізатору. Синтаксичний аналізатор — це частина, яка призначає значення лексемам на основі їх контексту. Парсер для SQLite генерується за допомогою генератора парсерів Lemon LALR3. Після того, як синтаксичний аналізатор збирає маркери в повні оператори SQL, він викликає генератор коду для створення коду віртуальної машини, який виконуватиме роботу, яку вимагають оператори SQL.
- Серверна частина містить B-дерево, кеш сторінок, інтерфейс ОС. База даних SQLite підтримується на диску за допомогою реалізації B-дерева, знайденого у вихідному файлі btree.c. Для кожної таблиці та індексу в базі даних використовується окреме B-дерево. Усі B-дерева зберігаються в одному дисковому файлі. Модуль B-дерева запитує інформацію з диска порціями фіксованого розміру. Кеш сторінок відповідає за читання, запис і кешування цих блоків. Щоб забезпечити переносимість між операційними системами

POSIX і Win32, SQLite використовує рівень абстракції для взаємодії з операційною системою.

- Аксесуари містять утиліти та тестовий код: SQLite надає деякі функції, пов’язані з утилітами, такі як розподіл пам’яті та процедури порівняння рядків без регістру, розташовані в util.c, більше половини загальної кодової бази SQLite присвячено тестуванню.

2.4. Знайомство з бібліотекою Pandas. Її призначення та функції

Pandas — це бібліотека Python, яка містить високорівневі структури даних та інструменти, які були створені, щоб допомогти Python програмістам виконувати потужний аналіз даних. Кінцева мета бібліотеки pandas — допомогти швидко знаходити інформацію в даних, де інформація визначається як базове значення. Розробку бібліотеки розпочав у 2008 році Вес МакКінні [4]. Зараз Pandas підтримується та активно розробляється різними організаціями та учасниками. Pandas спочатку було розроблено з урахуванням фінансів, зокрема з її можливостями щодо маніпулювання даними часових рядів та обробки історичної інформації про акції. Обробка фінансової інформації пов’язана з багатьма проблемами, серед яких:

- відображення даних про безпеку, як-от курс акцій, коли вони змінюються з часом;
- зіставлення вимірювань кількох потоків даних в однаковий час;
- визначення взаємозв'язку (кореляції) двох або більше потоків даних;
- представлення часу та дат як першокласних об’єктів;
- перетворення періоду вибірок даних у збільшення або зменшення.

Щоб виконати таку обробку, потрібен був інструмент, який би дозволяв отримувати, індексувати, очищати та охайно змінювати форму, об’єднувати, нарізати та виконувати різні аналізи як одновимірних, так і багатовимірних даних, у тому числі гетерогенних [4] даних, які автоматично вирівнюються вздовж набору

загальних індексних міток. Саме тут на допомогу приходить бібліотека Pandas, створена з багатьма корисними та потужними функціями, як-от:

- швидкі й ефективні об’єкти Series і DataFrame для обробки даних із вбудованим індексуванням;
- інтелектуальне вирівнювання даних за допомогою індексів і міток;
- інтегрована обробка відсутніх даних;
- засоби для перетворення безладних даних у впорядковані дані (прибирання);
- вбудовані інструменти для читання та запису даних між структурами даних у пам’яті та файлами, базами даних і веб-службами;
- можливість обробки даних, що зберігаються в багатьох поширених форматах, таких як CSV, Excel, HDF5 і JSON;
- гнучке змінення форми та зведення наборів даних;
- інтелектуальне нарізання на основі міток, фантастичне індексування та піднабір великих наборів даних;
- стовпці можна вставляти та видаляти зі структур даних для зміни розміру;
- агрегування або перетворення даних за допомогою потужного засобу групування даних для комбінування наборів даних із застосуванням розділення;
- високопродуктивне злиття та об’єднання наборів даних;
- ієрархічна індексація, що полегшує роботу з даними великої розмірності в структурі даних меншої розмірності;
- широкі функції для даних часових рядів, включаючи створення діапазону дат і перетворення частоти, статистику рухомого вікна, лінійну регресію рухомого вікна, зміщення дати та відставання;
- високо оптимізований для продуктивності, з критичними шляхами коду, написаними на Python або C.

Надійний набір функцій у поєднанні з безперебійною інтеграцією з Python та іншими інструментами в екосистемі Python забезпечив широке застосування Pandas у багатьох сферах. Він використовується в багатьох академічних і комерційних областях, включаючи фінанси, нейронауки, економіку, статистику, рекламу та веб-аналітику. Він став одним із найпопулярніших інструментів для науковців із обробки даних для представлення даних для маніпулювання та аналізу. Python довгий час був винятковим для обробки та підготовки даних, але меншою мірою для аналізу та моделювання даних. Pandas допомагає заповнити цю прогалину, дозволяючи виконувати весь робочий процес аналізу даних у Python без необхідності переходу на більш специфічну мову, таку як R. Це дуже важливо, оскільки ті, хто знайомі з Python, більш узагальненою мовою програмування, ніж R (більш статистичний пакет), отримує багато можливостей представлення даних і маніпулювання R, залишаючись повністю в неймовірно багатій екосистемі Python. У поєднанні з IPython, блокнотами Jupyter та багатьма іншими бібліотеками середовище для виконання аналізу даних у Python вирізняється продуктивністю та можливістю співпраці порівняно з багатьма іншими інструментами. Це призвело до широкого використання бібліотеки Pandas багатьма користувачами в багатьох галузях.

2.4.1. Маніпулювання даними, аналіз, наука та бібліотека Pandas

Ми живемо у світі, у якому щодня створюється та зберігається величезна кількість даних. Ці дані надходять із безлічі інформаційних систем, пристроїв і датчиків. Майже все, що ми робимо, і предмети, які ми для цього використовуємо, створюють дані, які можна або збирати. Це, у поєднанні з постійним зниженням вартості зберігання, зробило збір і зберігання навіть найтривіальніших даних ефективним. Це призвело до накопичення величезних обсягів даних, готових до доступу. Але ці дані розкидані по всьому кіберпростору, і насправді їх не можна назвати інформацією. Це, як правило, колекція записів подій, будь то фінансових, нашої взаємодії з соціальними мережами або особистого монітора здоров’я, який

відстежує серцебиття протягом дня. Ці дані зберігаються в усіх видах форматів, розташовані в розкиданих місцях, і поза їх необробленої природи дійсно дають багато розуміння.

Логічно загальний процес можна розділити на три основні дисципліни:

- маніпулювання даними;
- аналіз даних;
- наука про дані.

Ці три дисципліни можуть і мають багато збігів. Де кожна закінчується і де починаються інші, можна тлумачити довго.

Дані поширюються по всій планеті. Зберігається в різних форматах. Вони мають найрізноманітніші рівні якості. Через це існує потреба в інструментах і процесах для об’єднання даних у форму, яку можна використовувати для прийняття рішень. Це вимагає багатьох різних завдань і можливостей від інструменту, який маніпулює даними під час підготовки до аналізу. Функції, необхідні для такого інструменту, включають:

- можливість програмування для повторного використання та спільного використання;
- доступ до даних із зовнішніх джерел;
- зберігання даних локально;
- індексація даних для ефективного пошуку;
- вирівнювання даних у різних наборах на основі атрибутів;
- об'єднання даних у різні набори;
- перетворення даних в інші представлення;
- очищення даних від сміття;
- ефективна обробка неправильних даних;
- групування даних у загальні кошики;
- агрегація даних подібних характеристик;
- застосування функцій для обчислення значення або виконання перетворень;

- запитуйте та нарізайте, щоб досліджувати частини цілого;
- перебудова в інші форми;
- моделювання окремих категорій даних, таких як категоріальні, неперервні, дискретні та часові ряди;
- перевибірка даних на різних частотах.

Існує багато інструментів для обробки даних. Кожен з них відрізняється підтримкою елементів у цьому списку, способом їх розгортання та використання користувачами. Ці інструменти включають реляційні бази даних (SQL Server, Oracle), електронні таблиці (Excel), системи обробки подій (такі як Spark) і більш загальні інструменти, такі як R і Pandas.

Аналіз даних – це процес створення значення з даних. Дані з кількісним значенням часто називають інформацією. Аналіз даних – це процес створення інформації з даних шляхом створення моделей даних і математики для пошуку закономірностей. Це часто накладається на маніпуляції даними, і відмінність між ними не завжди чітка. Багато інструментів маніпулювання даними також містять функції аналізу, а інструменти аналізу даних часто надають можливості маніпулювання даними.

Наука про дані – це процес використання статистики та процесів аналізу даних для створення розуміння явищ у даних. Наука про дані зазвичай починається з інформації та застосовує до неї більш складний предметний аналіз. Ці домени охоплюють багато сфер, таких як математика, статистика, інформатика, інформатика, машинне навчання, класифікація, кластерний аналіз, аналіз даних, бази даних і візуалізація. Наука про дані є багатодисциплінарною. Її методи аналізу домену часто дуже різні та специфічні для конкретного домену.

2.4.2. Типи даних та змінних у Pandas

Працюючи з сирими даними, можна зіткнутися з кількома широкими категоріями даних, які потрібно буде ввести в структури даних Pandas. Їх важливо розуміти, оскільки інструменти, необхідні для роботи з кожним типом,

відрізняються. Pandas за своєю суттю використовується для маніпулювання структурованими даними, але надає кілька інструментів для полегшення перетворення неструктурованих даних у засоби, якими ми можемо маніпулювати.

Структуровані дані — це будь-який тип даних, організований як фіксовані поля в записі або файлі, наприклад дані в реляційних базах даних і електронних таблицях.

Структуровані дані залежать від моделі даних, яка є визначеною організацією та значенням даних, а часто й тим, як дані мають оброблятися. Це включає вказівку типу даних (ціле число, число з плаваючою точкою, рядок тощо) і будь-які обмеження на дані, такі як кількість символів, максимальне та мінімальне значення або обмеження на певний набір значень. Структуровані дані — це тип даних, для використання яких розроблено Pandas. Як видно спочатку з Series, а потім з DataFrame, Pandas організовує структуровані дані в один або більше стовпців даних, кожен з яких має один певний тип даних, а потім серію з нуля або більше рядків даних.

Неструктуровані дані — це дані, які не мають певної організації та не розбиваються на строго визначені стовпці певних типів. Вони можуть складатися з багатьох типів інформації, наприклад фотографій і графічних зображень, відео, потокових даних датчиків, веб-сторінок, PDF-файлів, презентацій PowerPoint, електронних листів, записів у блогах, вікі-сайтів і документів для обробки текстів. Хоча бібліотека Pandas не маніпулює неструктурованими даними напряму, вона надає низку можливостей для отримання структурованих даних із неструктурованих джерел. Як конкретний приклад, Pandas має інструменти для отримання веб-сторінок і витягування певних частин вмісту в DataFrame.

Напівструктуровані дані вписуються між неструктурованими. Цей тип даних можна вважати типом структурованих даних, але йому бракує суворої структури моделі даних. JSON — це форма напівструктурованих даних. Хоча хороший JSON матиме визначений формат, немає конкретної схеми для даних, яка завжди суворо дотримується. Значну частину часу дані будуть у повторюваному шаблоні, який можна легко перетворити на структуровані типи даних, такі як Pandas DataFrame,

але процес може потребувати певних вказівок, щоб указати або примусово задати типи даних.

Під час моделювання даних у Pandas моделюється одна чи декілька змінних і шукається статистичне значення серед значень або кількох змінних. Це визначення змінної не в значенні змінної в мові програмування, а є однією зі статистичних змінних. Змінна — це будь-яка характеристика, число або величина, яку можна виміряти або порахувати. Змінна називається так тому, що значення може відрізнятися між одиницями даних у сукупності та може змінюватися в значенні з часом. Прикладами змінних є вартість акцій, вік, стать, доходи та витрати від бізнесу, країна народження, капітальні витрати, класні оцінки, колір очей і тип транспортного засобу. Існує кілька широких типів статистичних змінних, які зустрічаються, використовуючи бібліотеку Pandas, а саме:

- категоріальні;
- безперервні;
- дискретні.

Категоріальна змінна [6] — це змінна, яка може приймати одне з обмеженої та зазвичай фіксованої кількості можливих значень. Кожне з можливих значень часто називають рівнем. Категоріальні змінні в Pandas представлені «Categoricals», типом даних Pandas, який відповідає категоріальним змінним у статистиці. Прикладами категоріальних змінних є стать, соціальний клас, групи крові, приналежність до країни, час спостереження або рейтинги, такі як шкали Лайкерта [6].

Безперервна змінна [6] – це змінна, яка може приймати нескінченно багато (незліченну кількість) значень. Спостереження можуть приймати будь-яке значення між певним набором дійсних чисел. Приклади безперервних змінних включають висоту, час і температуру. Безперервні змінні в Pandas представлені або типами з плаваючою точкою, або цілими числами (рідними для Python), як правило, у колекціях, які представляють кілька вибірок конкретної змінної.

Дискретна змінна – це змінна, значення якої базуються на підрахунку з набору різних цілих значень. Дискретна змінна не може бути дробовим значенням між будь-якими двома змінними. Приклади дискретних змінних включають кількість

зареєстрованих автомобілів, кількість місць розташування бізнесу або кількість дітей у сім'ї, усі з яких вимірюють цілі одиниці (наприклад, 1, 2 або 3 дітей). Дискретні змінні зазвичай представлені в Pandas цілими числами (або іноді плаваючими числами), знову ж таки зазвичай у колекціях двох або більше вибірок змінної.

Окремо необхідно згадати про дані часових рядів, які також присутні у Pandas. Дані часових рядів [6] є першокласною сутністю в Pandas. Час додає важливий додатковий вимір до зразків змінних у Pandas. Часто змінні не залежать від часу їх вибірки; тобто час, коли вони беруть пробу, не має значення. Але в багатьох випадках вони є.

Часовий ряд утворює вибірку дискретної змінної за певні проміжки часу, де спостереження мають природний часовий порядок. Стохастична модель для часового ряду, як правило, відображатиме той факт, що спостереження, близькі одне до одного в часі, будуть більш тісно пов'язані, ніж спостереження, які знаходяться далі одне від одного. Моделі часових рядів часто використовують природне одностороннє впорядкування часу, щоб значення для даного періоду виражалися як похідні певним чином із минулих значень, а не з майбутніх значень. Поширеним сценарієм у бібліотеці Pandas є фінансові дані, де змінна представляє вартість акцій, оскільки вона змінюється через рівні проміжки часу протягом дня. Нам часто необхідно визначити швидкість зміни ціни через певні проміжки часу. Або також виникає нобхідність співвіднести ціни кількох акцій через певні проміжки часу.

2.4.3. Процес аналізу даних в Pandas

Пропонований процес буде називатися процесом даних і представлений на рисунку 2.7. Цей процес встановлює основу для визначення логічних кроків, які виконуються під час роботи з даними. Наразі коротко розглянемо кожен із цих кроків у процесі та деякі завдання, які ми можемо виконувати за допомогою Pandas.

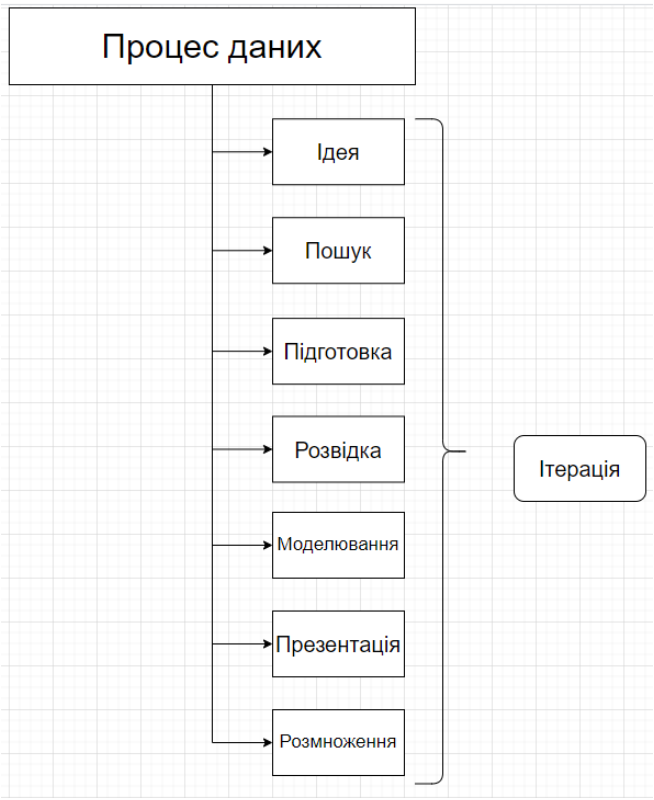

Рис. 2.7. Процес аналізу даних

Перший крок у будь-якій проблемі з даними – це визначити, що саме ви хочете з'ясувати. Це називається ідеєю, тобто придумати ідею того, що ми хочемо зробити та довести. Ідея зазвичай пов'язана з висуванням гіпотез щодо закономірностей у даних, які можна використовувати для прийняття розумних рішень. Ці рішення часто приймаються в контексті бізнесу, але також і в інших дисциплінах, таких як наука та дослідження. Зараз у моді розуміння операцій бізнесу, оскільки на розумінні даних часто потрібно заробити величезні суми грошей. Але яке рішення ми зазвичай хочемо прийняти? Нижче наведено кілька запитань, на які зазвичай ставлять відповіді:

- чому щось сталося;
- чи можемо ми передбачити майбутнє, використовуючи історичні дані;
- як можна оптимізувати роботу в майбутньому?

Цей список аж ніяк не є вичерпним, але він охоплює значний відсоток причин, чому будь-хто вдається до цих зусиль. Щоб отримати відповіді на ці запитання,

необхідно зібрати та зрозуміти дані, що стосуються проблеми. Це передбачає визначення того, які дані будуть досліджені, яка користь від дослідження, як дані будуть отримані, які критерії успіху та як інформація буде врешті передана. Pandas сама по собі не надає інструментів для допомоги в ідеях.

Коли у вас є ідея, ви повинні знайти дані, щоб спробувати підтвердити свою гіпотезу. Ці дані можуть надходити з певної організації або від зовнішніх постачальників даних. Ці дані зазвичай надаються як архівні дані або можуть надаватися в режимі реального часу (хоча Pandas не дуже відомий як інструмент обробки даних у режимі реального часу). Дані часто є дуже необробленими, навіть якщо отримані з джерел даних, які ви створили або з організації. Необробленість означає, що дані можуть бути неорганізованими, мати різні формати та бути помилковими; щодо підтримки аналізу, він може бути неповним і потребувати ручного розширення. У світі є багато безкоштовних даних. Багато даних не є безкоштовними, а їх отримання насправді коштує значних грошей. Деякі доступні у вільному доступі за допомогою загальнодоступних API, а інші – за підпискою. Дані, за які ви платите, часто чистіші, але це не завжди так. У будь-якому випадку Pandas надає надійний і простий у використанні набір інструментів для отримання даних із різних джерел, які можуть бути у багатьох різних форматах. Pandas також дає нам можливість не тільки отримувати дані, але й надавати початкову структуру даних за допомогою структур даних Pandas без необхідності вручну створювати складне кодування, яке може знадобитися в інших інструментах або мовах програмування.

Під час підготовки необроблені дані готуються до дослідження. Ця підготовка часто є дуже цікавим процесом. Дуже часто буває так, що дані з пов’язані з різними проблемами, пов’язаними з якістю. Можна витратити дуже багато часу на вирішення цих проблем якості, і часто це дуже нетривіальна кількість часу. Чому? Ну, є кілька причин:

- дані просто не корекнті;
- частини набору даних відсутні;
- дані не представлено з використанням вимірювань, які відповідають нашому аналізу;

- дані представлені у форматах, незручних для аналізу;
- рівень деталізації даних не підходить для аналізу;
- не всі потрібні поля доступні з одного джерела;
- відображення даних залежить від постачальника.

Підготовчий процес спрямований на вирішення цих питань. Pandas надає багато чудових можливостей для підготовки даних, які часто називають прибиранням даних. Ці засоби включають інтелектуальні засоби обробки відсутніх даних, перетворення типів даних, використання перетворення формату, зміну частоти вимірювань, об'єднання даних із кількох наборів даних, відображення/перетворення символів у спільні представлення та групування даних тощо.

Дослідження передбачає можливість інтерактивного розділення даних, щоб спробувати швидко зробити відкриття. Дослідження може включати різні завдання, такі як:

- вивчення того, як змінні співвідносяться одна з одною;
- визначення способу розподілу даних;
- пошук і виключення викидів;
- створення швидких візуалізацій;
- швидке створення нових представлень даних або моделей для використання в більш постійних і детальних процесах моделювання.

Дослідження є однією з сильних сторін бібліотеки Pandas. Хоча дослідження можна виконувати більшістю мов програмування, кожна з них має свій власний рівень формальності — скільки не дослідницьких зусиль потрібно докласти, перш ніж фактично дійти до відкриттів. При використанні з циклом читання-оцінки-друку у IPython та/або Jupyter Pandas створює дослідницьке середовище, яке майже не формальне. Виразність синтаксису Pandas дозволяє коротко описувати складні конструкції маніпулювання даними, а результат кожної дії, яку ви виконуєте з даними, негайно представляється для перевірки. Це дозволяє швидко визначити дію, яку ви щойно виконали, без необхідності перекомпільовувати та повністю запускати програми.

На етапі моделювання ми формалізуємо свої відкриття, знайдені під час дослідження, у чітке пояснення кроків і структур даних, необхідних для досягнення бажаного значення, яке міститься у наших даних. Це модель, поєднання обох структур даних, а також кроків у коді, щоб отримати від необроблених даних інформацію та висновки. Процес моделювання є ітеративним, коли шляхом дослідження даних ми вибираємо змінні, необхідні для підтримки аналізу, організовуємо змінні для введення в аналітичні процеси, виконуємо модель і визначаємо, наскільки добре модель підтримує наші вихідні припущення. Цей процес може включати формальне моделювання структури даних, але також може поєднувати методи з різних аналітичних областей, таких як (і не обмежуючись ними) статистика, машинне навчання та дослідження операцій. Щоб полегшити це, Pandas надає широкі засоби моделювання даних. Саме на цьому етапі ми більше переходимо від дослідження наших даних до формалізації моделі даних в об’єктах DataFrame та забезпечення стислості процесів для створення цих моделей. Крім того, працюючи на Python, ми можемо використовувати його повну потужність для створення програм для автоматизації процесу від початку до кінця. Створені нами моделі є виконуваними. З аналітичної точки зору, Pandas надає кілька можливостей, зокрема інтегровану підтримку описової статистики, яка може привести нас до бажаної мети для багатьох типів проблем. А оскільки Pandas базується на Python, якщо нам потрібні більш розширені аналітичні можливості, її дуже легко інтегрувати з іншими частинами великого наукового середовища Python.

Передостаннім кроком процесу є представлення наших висновків іншим, як правило, у формі звіту чи презентації. Можливо ми захочемо створити переконливе та докладне пояснення свого рішення? Це часто можна зробити за допомогою різних інструментів для побудови графіків в Python і створення презентації вручну. Блокноти Jupyter — потужний інструмент для створення презентацій для аналізу з бібліотекою Pandas. Ці блокноти надають засоби як для виконання коду, так і для надання широких можливостей уцінки для анотацій і опису виконання в кількох точках програми. Їх можна використовувати для створення дуже ефективних

виконуваних презентацій, візуально насичених фрагментами коду, стилізованим текстом і графікою.

Важливою частиною дослідження є обмін та забезпечення відтворюваності нашого дослідження. Часто кажуть, що якщо інші дослідники не можуть відтворити наш експеримент і його результати, то ми нічого не довели. На щастя для нас, використовуючи Pandas і Python, ми зможемо легко зробити свій аналіз відтворюваним. Це можна зробити, поділившись кодом Python, який керує нашим кодом Pandas. Блокноти Jupyter також забезпечують зручний спосіб упаковки коду та програми в засіб, яким можна легко поділитися з будь-ким, хто встановив Jupyter Notebook. І в Інтернеті є багато безкоштовних і безпечних сайтів для обміну, які дозволяють створювати або розгортати наші блокноти Jupyter для спільного використання.

2.5. Середовище розробки PyCharm. Основні можливості

2.5.1. Загальний огляд середовища розробки

Щоб зрозуміти «Що таке PyCharm?» і «Для чого використовується PyCharm?», ми повинні спочатку відповісти на запитання «Що таке інтегроване середовище розробки?»

Інтегроване середовище розробки (IDE) складається з редактора та компілятора, які ми використовуємо для написання та компіляції програм. Він має комбінацію функцій, необхідних для розробки програмного забезпечення.

Наявність IDE значно полегшує процес розробки та програмування. Він інтерпретує те, що ми вводимо, і пропонує відповідне ключове слово для вставки. Ми можемо розрізнити клас і метод, оскільки IDE надає їм різні кольори. IDE також надає різні кольори для правильних і неправильних ключових слів. Якщо ми пишемо неправильне ключове слово, він намагається передбачити ключове слово, яке ми маємо намір написати, і автоматично завершує його.

До основних причин використання IDE для розробки можна віднести:

- IDE складається з вікна текстового редактора, де ми можемо писати наші програми;
- воно складається з вікна редактора проекту, де ми зберігаємо всі необхідні файли проекту програмного забезпечення;
- ми можемо надавати різні вхідні дані та перевіряти ефективність нашої програми, перевіряючи вихідні дані, які ми отримуємо у вікні виводу;
- у разі виникнення будь-якої помилки IDE показує попередження та пропозиції у вікні виводу, щоб ми могли її вирішити;
- IDE містить стелаж із модулями та пакетами в одному місці, що допомагає додавати функції в наші програми;
- Воно допомагає підвищити ефективність створення програмного забезпечення.

PyCharm — це гібридна платформа, розроблена JetBrains як IDE для Python. Він зазвичай використовується для розробки додатків Python. Деякі організації, такі як Twitter, Facebook, Amazon і Pinterest, використовують PyCharm як свою IDE Python. Він підтримує дві версії: v2.x і v3.x.

Ми можемо запускати PyCharm на Windows, Linux або Mac OS. Крім того, він містить модулі та пакети, які допомагають програмістам розробляти програмне забезпечення за допомогою Python за менший час і з мінімальними зусиллями. Крім того, його також можна налаштувати відповідно до вимог розробників.

2.5.3. Тестова програма для отримання даних про якість повітря

Розглянемо тестову програму, після компіляції та запуску якої, ми отримаємо файлу типу JSON з необхідними нам даними про якість повітря у місті Київ. Перш за все, імпортуємо необхідні модулі для відправки запитів.

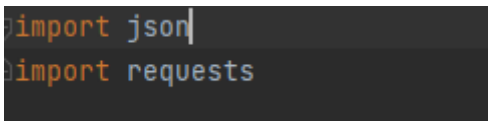

Рис. 2.8. Імпорт пакетів необхідних для відправки запиту

Далі оголосимо функцію «request», яка буде виконувати запит та записувати це у файл.

```
def request(url, headers, params):
    response = requests.request("GET", url, headers=headers, params=params)
    response = response.json()

    return response
```

Рис. 2.9. Функція request

Параметрами цієї функції, як видно з рисунку 2.9 є три змінні: url, headers та params. Змінна url – це посилання на API, яке містить в собі дані про якість повітря. Змінні headers та params – це змінні, які містять в собі дані необхідні для авторизації, а саме API ключ, заголовки та довжину і широту міста, для якого ми шукаємо дані про якість повітря. Повертає функція JSON стрічку з необхідними параметрами.

Для того, щоб отримати файл з кінцевими даними, нам необхідно реалізувати функцію write_to_file, яка ортимає на вхід JSON стрічку з параметрами якості повітря, які ми отримали у функції request та запише їх у файл.

```
def write_to_file(data):
    with open("sample.json", "w") as outfile:
        json.dump(data, outfile)
```

Рис. 2.10. Реалізація функції «write_to_file»

Після закінчення роботи цієї функції, створиться файл, який буде мати назву sampe.json, та буде містити необхідні нам дані про якість повітря у Києві.

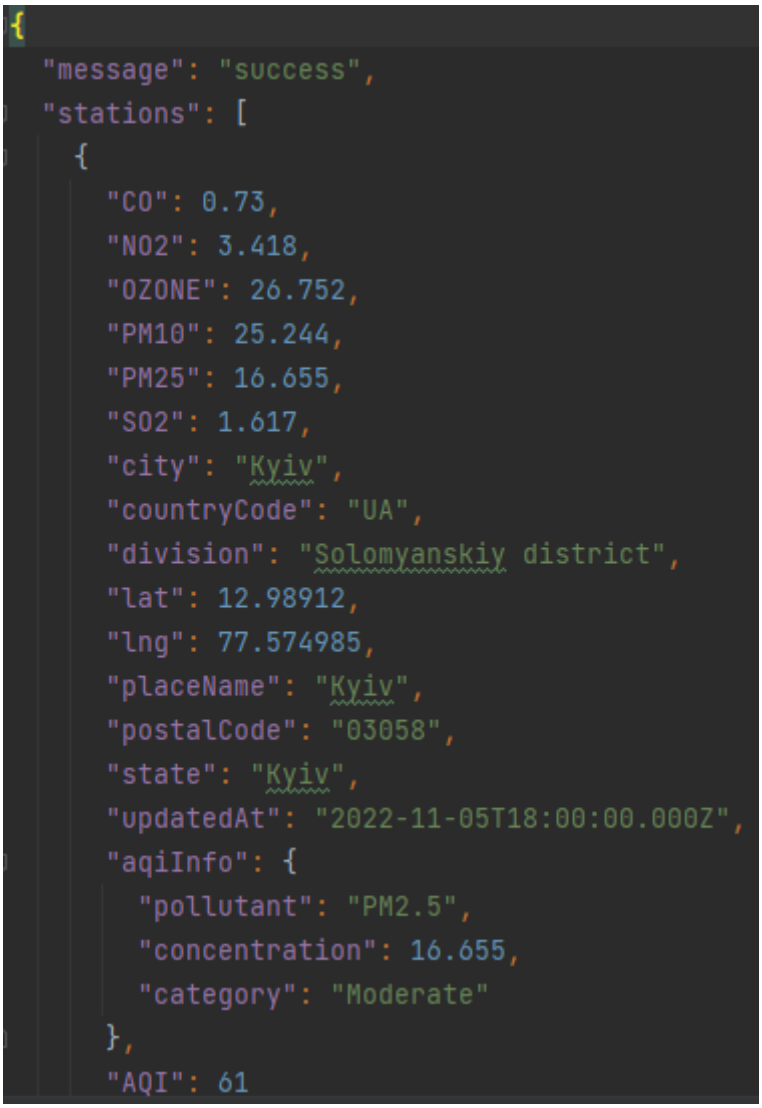

Рис. 2.11. Кінцевий файл «sample.json» з вихідними даними

2.6. Висновки до розділу 2

У цьому розділі були розглянуті основи роботи алгоритму «випадкового лісу». З наведених тез та аргументів можна зробити висновок, що саме цей алгоритм модна використовувати для найбільш точного прогнозування необхідної нам інформації. Також у розділі 2 були розглянуті основні інструменти, які будуть використані для реалізації системи контролю якості повітря на основі даних, які

будуть отримані в реальному часі. Тут були описані необхідні бібліотеки, які мають в собі реалізацію математичних операцій для роботи з великими обсягами даних, великий фреймворк Django, який буде слугувати для створення веб-сайту для відображення результатів роботи алгоритму та пакет Scikit-learn. Також було приведено приклад роботи тестової програми, де було описано принцип спілкування та надсилання запитів до стороннього API, щоб завжди отримувати актуальні дані про якість повітря у обраному місті.

З цього розділу можна зробити висновок, що обрані інструменти є найбільш оптимальними та не потребують великих ресурсів та часу, який потрібно витратити на їх освоєння.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ АНАЛІТИЧНОГО ПРОГНОЗУВАННЯ ЗАБРУДНЕННЯ ПОВІТРЯ НА ОСНОВІ ДАНИХ, ОТРИМАНИХ В РЕАЛЬНОМУ ЧАСІ

3.1. Створення Django проекту для інтеграції алгоритму «Випадкового лісу». Його структура

Для створення Django проекту, нам необхідно відкрити командний рядок, та вписати команду «django-admin startproject air_conditions». Ця команда відповідає за створення каталогу з всіма необхідними для роботи файлами та теками.

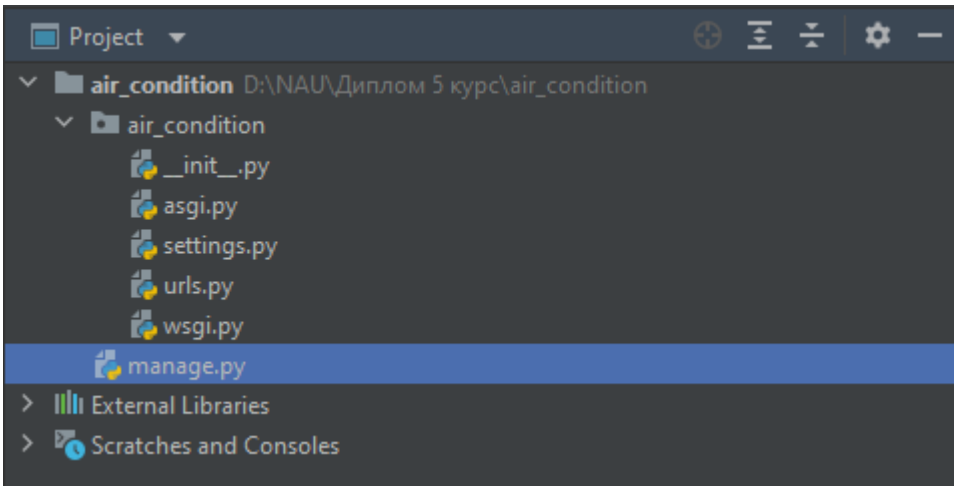

Рис. 3.1. Структура проекту «air_condition»

У наведеній на рисунку 3.1. структурі, ми бачимо 6 файлів, які генеруються автоматично. Файл «__init__.py» потрібен для оголошення класів або пакетів, які можуть бути використані пізніше у роботі.

Файли «asgi.py» та «wsgi.py» відповідають за конфігурацію для нашого проекту. У них визначені глобальні змінні для роботи серверної частини Django. Наступний файл «settings.py» необхідний для явного оголошення в ньому додатків, які ми будемо створювати для цього проекту, шаблонів, бази даних та змінних оточення для проекту. Файл «urls.py» служить для оголошення посилань на сторінки, які будуть відображатись в браузері. Найголовніший файл «manage.py» відповідає за операції старту серверу, на якому ми будемо запускати наш проект, створення додатків для проекту та інше.

3.2. Додаток «cities». Структура додатку та реалізація логіки роботи

3.2.1. Створення додатку «cities». Його вигляд

Для створення додатку «cities», який в подальшому буде слугувати нам як основна частина робочої програми, де відбуваються всі обчислення та логіка відбувається за допомогою команди «python manage.py startapp cities».

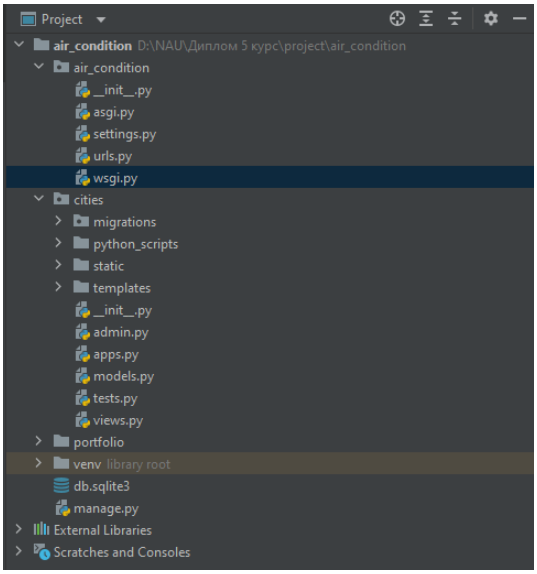

Рис 3.2. Структура проекту після створення додатку

Як видно на рисунку 3.2., у каталозі з’явилась нова папка з назвою нашого додатку. Він містить у собі файли, які відповідають за адмін панель, наповнення бази даних, опис функцій, які будуть використовуватись для запуску алгоритму та відображення результатів на веб-сторінці.

Перш за все, необхідно створити модель, яку буде записана у базу даних «SQLite3» та буде містити у собі міста, для яких буде перевірятись якість повітря. Для цього нам необхідно відкрити файл «models.py» та створити клас під назвою «City», де будуть описані основні поля таблиці.

```python
from django.db import models


class City(models.Model):
    title = models.CharField(max_length=100)
    lat = models.FloatField()
    lon = models.FloatField()

    def __str__(self):
        return self.title
```

Рис. 3.3. Ініціалізація та реалізація класу City()

Після того, як ми реалізували клас «City», обов’язково необхідно зробити міграцію у базу даних за допомогою команди «python manage.py makemigrations». Це потрібно для того, щоб у подальшому ми змогли отримувати назви міст та їх координати для успішного виконання запиту у сторонній API сервіс та отримання даних про якість повітря у кожному місті. Після виконання цієї команди, моделі буде відскановано та порівняно з версіями, які наразі містяться у файлах міграції, а потім буде виписано новий набір міграцій. Обов’язково потрібно прочитати вихідні дані, щоб побачити, що команда «makemigrations» вважає, що ви змінили – це не ідеально, і для складних змін він може не виявляти того, що ви очікуєте. Отримавши нові файли міграції, ми повинні застосувати їх до своєї бази даних, щоб переконатися, що вони працюють належним чином. Для цього, потрібно запустити

команду «python manage.py migrate». Щойно міграцію буде застосовано, ми зафіксовуємо міграцію, і моделі змінюватимуться у нашій системі контролю версій у вигляді єдиного коміту – таким чином, коли інші розробники перевірять код, вони отримають обидві зміни наших моделей і водночас супутню міграцію. Також важливим кроком є те, що необхідно зареєструвати нашу модель у проекті Django. Для цього, перейдемо у файл «admin.py» та впишемо код, який показаний на рисунку 3.4. Це необхідно для доступу до адмін панелі нашого проекту.

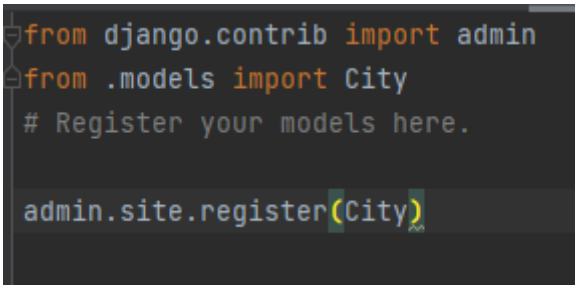

Рис. 3.4. Реєстрація моделі «cities» у проекті

3.2.2. Створення шаблонів сторінок

Шаблони є третьою і найважливішою частиною структури Django MVT. Шаблон у Django в основному написаний на HTML, CSS і Javascript у файлі .html. Фреймворк Django ефективно обробляє та генерує динамічно веб-сторінки HTML, видимі для кінцевого користувача. Django в основному працює з бекендом, тому, щоб забезпечити інтерфейс і надати макет нашого веб-сайту, ми використовуємо шаблони. Існує два способи додавання шаблону на наш веб-сайт залежно від наших потреб.

Ми можемо використовувати єдиний каталог шаблонів, який буде розподілено по всьому проекту.

Для кожної програми нашого проекту ми можемо створити окремий каталог шаблонів.

Для нашого поточного проекту ми створимо єдиний каталог шаблонів, який буде розподілено по всьому проекту для простоти. Шаблони на рівні програми

зазвичай використовуються у великих проектах або у випадку, якщо ми хочемо надати інший макет для кожного компонента нашої веб-сторінки. На рисунку 3.5 зображена структура теки з шаблонами.

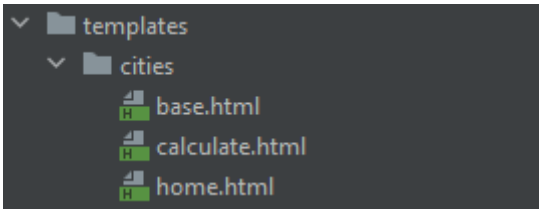

Рис. 3.5. Структура папки з шаблонами

З рисунку 3.5 видно, що я створив 3 файли шаблонів. Один з них це базовий шаблон, та 2 шаблони, які наслідуються від базового та описують окрему сторінку. Базовий шаблон необхідний більше для зручності та чистоти коду, оскільки з точки зору використання, більш зручно не описувати базову структуру сторінки кожен раз, а мати один уніфікований шаблон, який містить у собі всі необхідні компоненти.

Перейдемо до наповнення шаблонів. Першим наповнимо базовий шаблон «base.html». Оскільки у проекті будуть використовуватись JavaScript, CSS та HTML – нам необхідно вказати це у самому шаблоні. Також, опишемо фон веб-сторінки та верхню панель, на якій буде розміщена назва нашої системи, посилання на API, яке було використане для збору даних та посилання на мій профіль у LinkedIn.

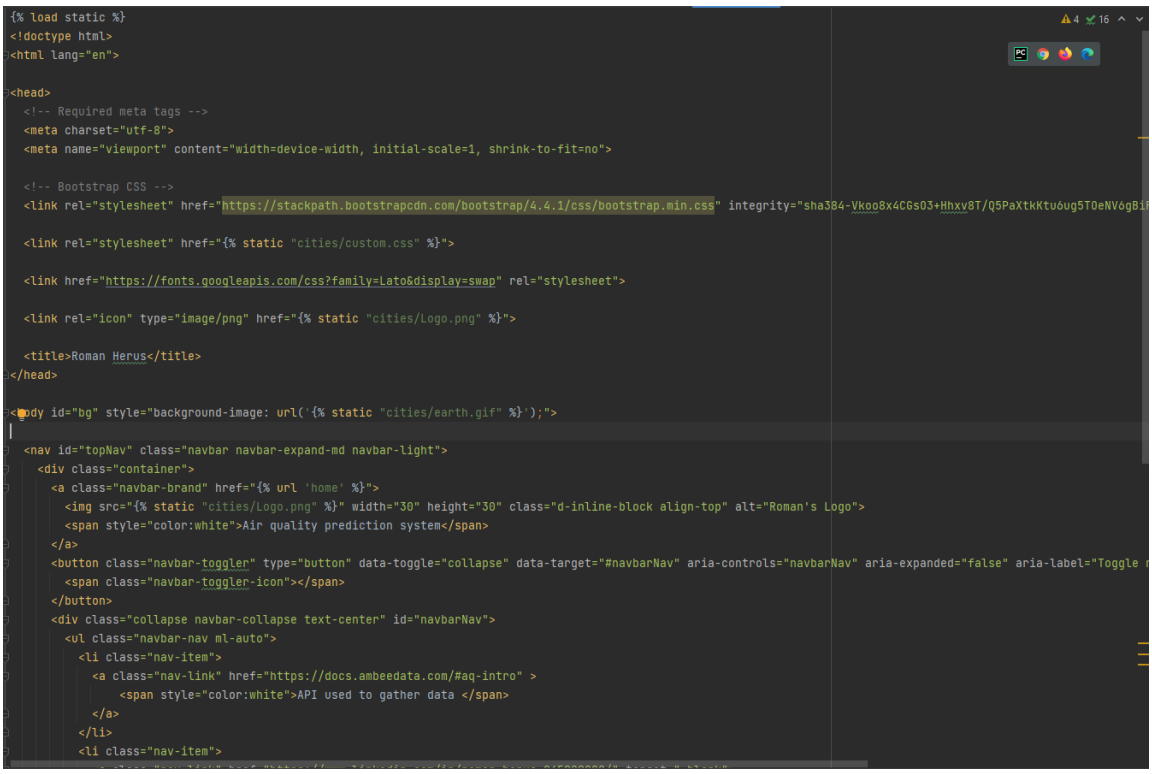

Рис. 3.6. Наповнення шаблону «base.html»

З рисунку 3.6 видно, що весь необхідний функціонал був описаний. Зверху документа явно вказано, що нам необхідно підключити всі файли, які розміщені у каталозі під назвою «static». Цей каталог необхідний для збереження рисунків, файлів або іншого контенту, який є статичним та ніколи не буде змінюватись користувачем.

Запустимо наш сервер та додаток. Для цього нам необхідно виконати команду «python manage.py runserver». Наш додаток запускається на локальному комп’ютері та доступний за локальною адресою з портом доступу 8000. На рисунку 3.7 видно, як виглядає перша версія нашого сайту. Наступні шаблони будуть використані для наповнення окремих сторінок, які будуть присутні у проекті.

Рис. 3.7. Вигляд базового шаблону у веб браузері

Також важливим аспектом є створення файлу «custom.css». Він необхідний для того, щоб елементи які ми будемо розміщувати на нашій сторінці мали гарний вигляд, а саме кольори елементів, їх позиції та рівень прозорості.

```
.navbar-nav .nav-link {
    color: rgba(0,0,0,1) !important;
}

.navbar-nav .nav-link:hover {
    color: rgba(0,0,0,.5) !important;
}

body {
    font-family: 'Lato', sans-serif;
    color: white;
}

#blogtitle {
    font-size: 6rem;
}

@media (min-width: 992px) {
    #hometext {
        font-size: 4rem;
    }
    #blogdetailtitle {
        font-size: 5rem;
    }
}

.grid {
    position: absolute;
    top: 400px;
    display: grid;
    column-gap: 20px;
    grid-template-columns: 20% 20% 20% 20% 20%;
}
```

Рис. 3.8. Наповнення файлу «custom.css»

Перейдемо до створення шаблону домашньої сторінки. Домашня сторінка – це веб-сторінка яку буде бачити користувач, щойно відвідавши наш сайт. За домашню сторінку відповідатиме шаблон під назвою «home.html». Тут ми опишемо текст, який буде бачити користувач при відвідуванні сайту та розмістимо п’ять плиток з назвами міст, для яких буде доступне прогнозування якості повітря.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
{% extends "cities/base.html" %}

{% load static %}

{% block content %}


<div class="row justify-content-center mb-3">
  <div class="col-md-9 text-center">
    <h1 id="hometext" class="font-weight-bold">Discover new opportunities for air quality forecasting</h1>
  </div>
</div>

<div class="grid">
    {% for city in cities %}
    <div class="box box2">{{ city.title }}
        <button onclick="location.href='{% url 'calculate' city=city.title %}'" class="btn btn-primary">Make a forecast</button> <hr>
    </div>
    {% endfor %}
</div>

{% endblock %}
</body>
</html>
```

Рис. 3.9. Файл шаблону головної сторінки

Після того, як ми створили цей шаблон, ми можемо переглянути, як він виглядає у веб браузері. Оскільки сервер вже було запущено, у нас немає потреби робити це ще раз або перезапускати його. Зміни одразу вступлять в силу після перезавантаження сторінки.

Рис. 3.10. Вигляд домашньої сторінки після створення її шаблону

Останнім шаблоном буде файл «calculate.html». Він необхідний для того, щоб відображати сторінку з результатом прогнозування якості повітря для конкретного міста. На цій сторінці пізніше буде розміщено графік, де будуть зображені результати роботи алгоритму.

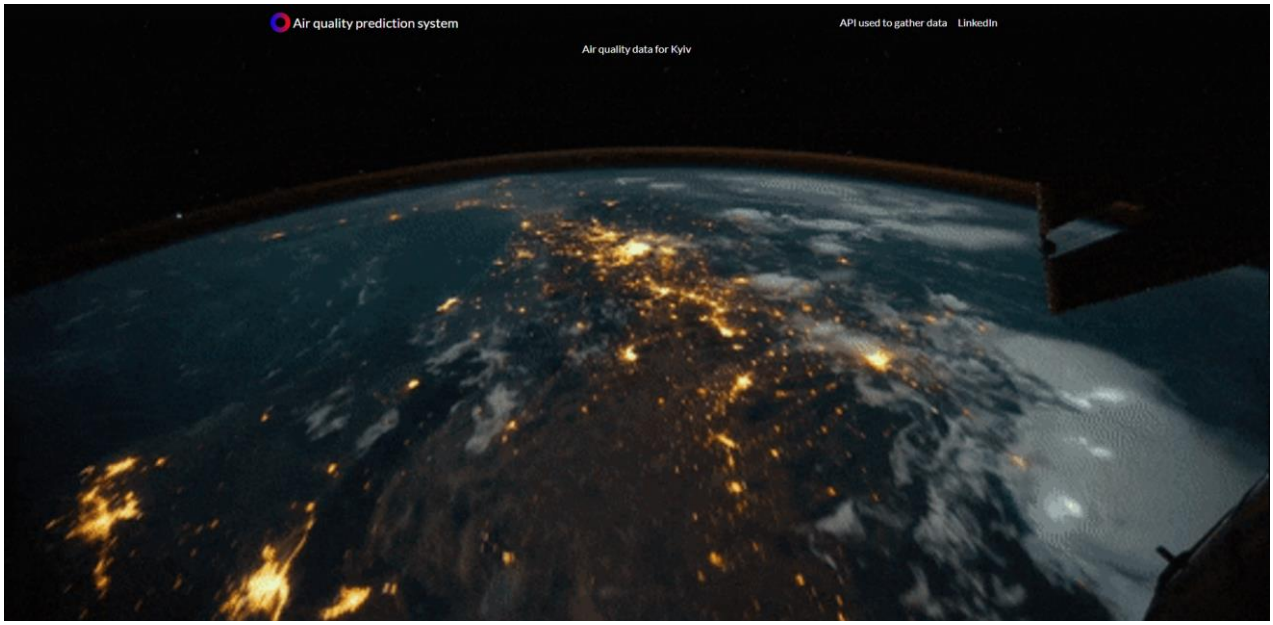

Рис. 3.11. Вигляд сторінки з результатами

3.3. Реалізація алгоритму «випадкового лісу» та передача даних з коду на веб сторінку

3.3.1. Логіка роботи алгоритму «випадкового лісу»

Для успішного прогнозування якості забруднення повітря, необхідно реалізувати логіку роботи самого алгоритму. Сама реалізація алгоритму виконана у бібліотеці sklearn. Ця бібліотека використовується для того, щоб унеможливити похибки, які можуть бути допущені при самостійній реалізації алгоритму, оскільки він є дуже комплексним.

Так як було сказано у розділі 2, алгоритм для коректної роботи алгоритму, спочатку потрібно реалізувати файл, де алгоритм на отриманих даних буде тренуватись. Для цього створимо файл «train.py» у директорії «python_scripts». У функції «train()» буде описана вся логіка тренування алгоритму, а саме зчитування даних про якість повітря отриманих в реальному часі з файлу, який буде створено після звертання до API, розподілення цих даних на набори даних та сам процес тренування.

```python
from sklearn.ensemble import RandomForestRegressor

column_names = ['Date', 'Nitrogen_Dioxide', 'Ozone', 'PM10_Particulate', 'PM2-5_Particulate']


def train(city):
    data_set = 'cities/python_scripts/air-quality-{0}-time-of-day.csv'.format(str(city).lower())

    data = pd.read_csv(data_set, sep=',', names=column_names)
    X = data.iloc[1:, 0].values
    Y = data.iloc[1:, 4].values
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

    print(type(X_train))

    X_train = X_train.reshape(-1, 1)
    X_test = X_test.reshape(-1, 1)

    regressor = RandomForestRegressor(n_estimators=4000)
    regressor.fit(X_train, Y_train)
    y_pred = regressor.predict(X_test)

    joblib.dump(regressor, 'cities/python_scripts/pollution_prediction.pkl')
```

Рис. 3.12. Реалізація функції для тренування алгоритму

З рисунку 3.12 видно, що дані беруться з файлу для конкретного міста, відбувається процес розподілення цих даних та тренування алгоритму. Опісля, результат тренування записується у файл з спеціальним розширенням, для прогнозування якості забруднення повітря.

Наступним кроком стане реалізація файлу «», де буде описана логіка прогнозування якості повітря. Тут ми будемо отримувати назву міста, для якого необхідно зробити прогноз, викликати функцію для прогнозування та записувати дані.

```python
import numpy as np
import joblib


def predict_pollution(city, option):
    tree_model = joblib.load('cities/python_scripts/pollution_prediction.pkl')
    date = [float(option)]
    date = np.asarray(date, dtype=np.float32)
    date = date.reshape(-1, 1)
    print(date)
    temp = tree_model.predict(date)[0]
    print("-" * 48)
    print("\nThe pollution is estimated to be: " + str(temp) + "\n")
    print("-" * 48)
    return str(temp)
```

Рис. 3.13. Реалізація функції алгоритму для прогнозування

3.3.2. Реалізація функцій у файлі «views» для передачі даних на веб сторінку

Як було вказано у розділі 2, Django використовує шаблон програмування MVT. Це означає, що повинні бути реалізовані спеціальні функції для кожної сторінки, які будуть містити у собі логіку для відображення необхідних нам даних на веб сторінці.

Першою функцією, яка буде реалізована у файлі «views» буде функція «», яка буде відповідати за рендер головної сторінки та передачу даних про міста на головну сторінку та у шаблони. Оскільки Django спрощує роботу з базою даних, то отримати об’єкт міста не є великою задачою. Повертає функція головну сторінку та дані про місто.

```python
# Create your views here.
def home(request):
    cities = City.objects.all()
    return render(request, 'cities/home.html', {'cities': cities})
```

Рис. 3.14. Реалізація функції «home»

Наступною і найголовнішою функцією буде функція «calculate». Вона буде відповідати за рендер сторінки з результатами роботи алгоритму, відправкою запиту на сторонній API для отримання даних про якість повітря у певному місті та їх обробкою.

```
def calculate(request, city):
    city_name = get_object_or_404(City, title=city)
    # call function
    url = "https://api.ambeedata.com/latest/by-lat-lng"
    querystring = {"lat": "12.9889055", "lng": "77.574044"}
    headers = {
        'x-api-key': "b6cf602aa6a13ad00663a99fc8149dc169564f9192fed6964d1baf266e4835f0",
        'Content-type': "application/json"
    }
    response = requests.request("GET", url, headers=headers, params=params)
    response = response.json()

    data = {}
    date = {
        1: datetime.now().strftime('%d%m%Y15'),
        2: (datetime.now() + dt.timedelta(days=1)).strftime('%d%m%Y15'),
        3: (datetime.now() + dt.timedelta(days=3)).strftime('%d%m%Y15'),
    }
    run.train(city_name)
    for i in date.values():
        value = pr.predict_pollution(city_name, i)
        datetime_object = datetime.strptime(i[:8], '%d%m%Y').date()
        print(datetime_object)
        data[str(datetime_object)] = value

    plt.plot(data.values(), data.keys(), color='red', marker='o')
    plt.title('Air pollution in the next 3 days for {0}'.format(city_name), fontsize=14)
    plt.xlabel('AIQ', fontsize=14)
    plt.ylabel('Date', fontsize=14)
    plt.grid(True)
    buffer = BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    image_png = buffer.getvalue()
    buffer.close()
    graphic = base64.b64encode(image_png)
    graphic = graphic.decode('utf-8')

    return render(request, 'cities/calculate.html', {'graphic': graphic})
```

Рис.3.15. Реалізація функції «calculate»

З рисунку 3.15 видно, що на вхід у функцію поступає два параметри «request» та «city». Перший параметр – це запит, який отримує сервер після того, як ми нажимаємо на кнопку «зробити прогноз» біля обраного міста. Другий – це назва міста, яке обирає користувач. Воно необхідно для того, щоб ми змогли відправити запит на сторонній API та отримати дані про якість повітря у цьому місті. Система сконструйована таким чином, що в незалежності від дати та часу, коли користувач робить запит, він буде отримувати прогноз якості повітря на 3 дні вперед. Ця логіка

описана в словнику «date», елементами якого є 3 дні, починаючи з моменту коли користувач робить запит. Після цього, запускається процес тренування алгоритму та процес прогнозування для кожної дати, яка була сформована кроком раніше.

Наступним завданням буде графічно відобразити прогнозовані. Для цього буде використана бібліотека «matplotlib». Принцип її роботи досить простий: на вхід подається два списки даних для двох осей – X та Y. Після цього створюються дві осі з підписами та розміщення необхідних точок з даними на графіку. Але, оскільки, кінцевим результатом роботи цієї бібліотеки є діалогове вікно з графіком, нам також необхідно реалізувати збереження цього графіку як потік байтів, для подальшого відображення на веб сторінці. Для цього будуть використані вбудовані у python методи перетворення зображення у набір байтів. Результатом роботи цієї функції є сторінка з відображеним графіком прогнозу якості повітря.

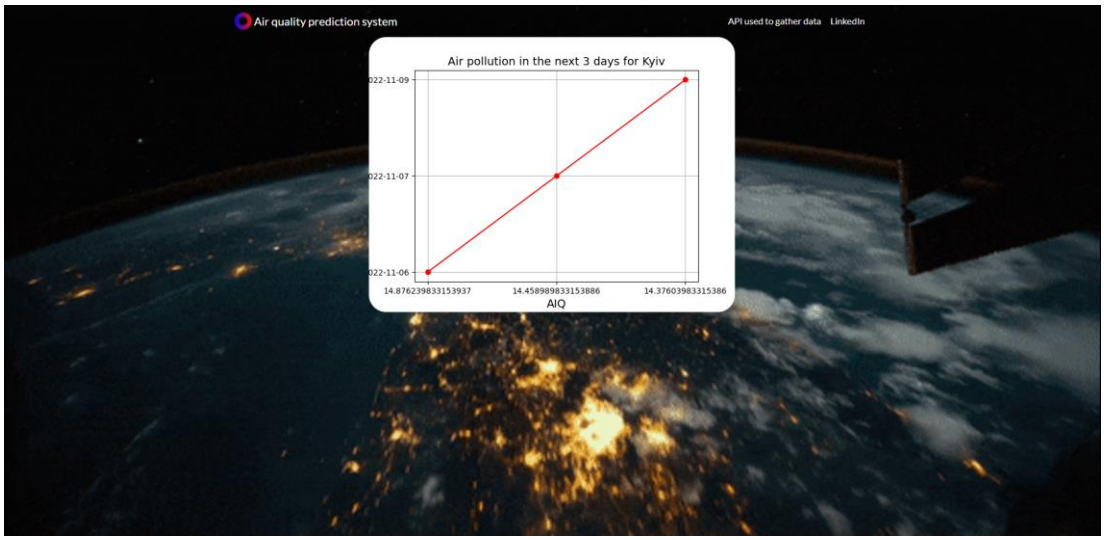

Рис. 3.16 Сторінка з графіком прогнозу повітря.

З рисунку 3.16 видно, що в результаті роботи функції «calculate» ми отримали графік з прогнозом якості повітря для міста Київ на період починаючи з 6 Листопада до 9 Листопада. Вісь X на графіку відображає значення AQI, а вісь Y – дату, для якої ці дані будуть актуальні.

3.4. Висновки до розділу 3

Основними етапами при розробці комп’ютерного пристрою контролю якості повітря є створення файлів, які будуть мати у собі логіку для тренування алгоритму «випадкового лісу» та створення Django додатку, який потрібен для відображення результатів прогнозування алгоритму. Для створення та налаштування алгоритму та Django додатку були створені такі функції та файли:

- функції train() і predict() для алгоритму;
- функції home() та calculate() для файлу views.py та клас City для файлу models.py, для того, щоб відобразити веб сторінки та створити таблицю у базі даних відповідно;
- файли шаблонів base.html, home.html та calculate.html для рендеру веб сторінок у браузері.

В результаті була створена система з прогнозування якості повітря у одному з п’яти наявних міст на три дні вперед, користувачі якої можуть слідкувати та оцінювати якість повітря у своєму місті та вживати відповідних мір для своєї безпеки.

ВИСНОВКИ

В процесі дослідження теми з прогнозуванням якості повітря у світі стало очевидно, що людство не використовує наявні технології на повну силу. У сучасному світі ми легко можемо отримати різні види прогнозів, в тому числі й погоди, але це не завжди є точним та не існує ніяких альтернативних варіантів з застосуванням технологій, де можна зменшити похибку та програмно дослідити шаблони поведінки.

Постійний контроль за якістю повітря – це важливий аспект сьогоденного життя, оскільки кількість викидів у атмосферу збільшується у геометричній прогресії та в подальшому негативно вплине на якість життя.

В процесі розробки були використані наступні технології:

- мова програмування python. Сучасна швидксіна мова програмування яка є наслідником C, довела свою простоту та універсальність;
- Бібліотека Django. Django — це високорівневий веб-фреймворк для Python, який заохочує швидку розробку та чистий, прагматичний дизайн;
- Бібліотека NumPy. NumPy — це основний пакет для наукових обчислень на Python;
- Бібліотека Pandas – бібліотека призначена для обробки великих обсягів даних та їх аналізу в подальшому.
- Пакет SKlearn. Sklearn — це надкорисна та надійна бібліотека для машинного навчання на Python. Він надає вибір ефективних інструментів для машинного навчання та статистичного моделювання, включаючи класифікацію, регресію та кластеризацію через узгоджений інтерфейс у Python.

Основними можливостями розробленого комп’ютерного пристрою контролю якості повітря є:

- тренування алгоритму «випадкового лісу» для збільшення точності прогнозованих результатів;
- прогнозування індексу забруднення повітря на три дні вперед;
- графічне відображення результатів роботи алгоритму на веб сторінці.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tan. Introduction to Data Mining / Tan, Steinbach, Kumar; Вид-во Pearson, 2005 – 792 p.
2. Charu C. Linear Algebra and Optimization for Machine Learning: A Textbook / Charu C. Aggarwal; Вид-во Springer, 2020 – 516p.
3. Mehmed K. Data Mining: Concepts, Models, Methods, and Algorithms / Mehmed Kantardzic; Вид-во Wiley-IEEE Press, 2019 – 672p.
4. Wil M. Process Mining: Data Science in Action / Wil M. P. van der Aalst; Вид-во Springer, 2018 – 486p.
5. Han J. Data Mining: Concepts and Techniques / Jiawei Han; Вид-во Morgan Kaufmann, 2022 – 762p.
6. Kumar N. Blockchain, Big Data and Machine Learning / Neeraj Kumar; Вид-во CRC Press, 2022 – 360p.
7. Liu B. Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data / Bing Liu; Вид-во Springer, 2013 – 644p.
8. Unhelker B. Applications of Artificial Intelligence and Machine Learning / Unhelker B., Mohan Pandey H., Raj G.; Вид-во Springer, 2022 – 1285p.