Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python»

08-54.БКР.031.00.000 ПЗ

Виконала: студентка 4 курсу, групи 2КІ-23мс
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Комп'ютерна інженерія»
Шалаган В.С. _____
Керівник: доцент кафедри ОТ, к.т.н.
Дудник О.В. _____
Рецензент: доцент кафедри ПЗ, к.т.н.
Романюк О.В. _____

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
« » _____ 2025 р.

Вінниця, ВНТУ — 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо–кваліфікаційний рівень: бакалавр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. Азаров О.Д.
«_21_» _березня_____ 2025 року

**З А В Д А Н Н Я**

**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ**

Шалаган Вікторії Сергіївні

1 Тема роботи: Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python, керівник роботи Дудник Олександр Вікторович, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від « 20 » _____березня_____ 2025 року №97.

2 Строк подання студентом роботи:____13.05.2025_____2025 року.

3 Вихідні дані до роботи: дані з агрегаторів авіаквитків, специфікація структури даних, умови валідації даних, протокол передачі даних SFTP, технічна документація на бібліотеки Python (pandas, paramiko, sqlalchemy), тестові датасети

4 Зміст розрахунково-пояснювальної записки: вступ, огляд сучасних технологій ETL та методів обробки авіаційних даних, розробка ETL-системи на Python для обробки даних з агрегаторів авіаквитків, реалізація валідації даних, тестування системи, візуалізація результатів обробки, висновки.

5 Перелік ілюстративного матеріалу: структури файлів.

6 Консультанти розділів кваліфікаційної роботи приведені в таблиці 1.

Таблиця 1 – Консультанти роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 - 4 | доц.каф. ОТ, к.т.н. Дудник О.В. | 21.03.25 | 13.05.25 |
| Нормоконтроль | ас. каф. ОТ Швець С.І. | 14.05.25 | 30.05.25 |

7 Дата видачі завдання: 21.03.2025 р.

8 Календарний план виконання кваліфікаційної роботи приведени таблиці 2.

Таблиця 2 — Календарний план

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання | Примітка |
|---|---|---|---|
| 1 | Формулювання мети, завдань та обґрунтування актуальності теми | 21.03.25 | Виконано |
| 2 | Аналіз методів побудови ETL-систем, огляд бібліотек та підходів | 24.03.25 — 28.03.25 | Виконано |
| 3 | Розробка ETL-системи обробки авіаційних даних на основі Python | 31.03.25 — 04.04.25 | Виконано |
| 4 | Інтеграція з SFTP-сервером, обробка CSV та формування структури БД | 07.04.25 — 11.04.25 | Виконано |
| 5 | Побудова DAG-графа в Apache Airflow | 14.04.25 — 18.04.25 | Виконано |
| 6 | Розробка модулів валідації даних та перевірка типових помилок | 21.04.25 — 25.04.25 | Виконано |
| 7 | Побудова візуалізацій результатів валідації | 28.04.25 — 02.05.25 | Виконано |
| 8 | Оформлення пояснювальної записки, додатків та ілюстрацій | 05.05.25 — 12.05.25 | Виконано |
| 9 | Підсумкова перевірка роботи та усунення недоліків | 13.05.25 | Виконано |

**Студентка** Вікторія ШАЛАГАН

**Керівник роботи** к.т.н., доц. Олександр ДУДНИК

# АНОТАЦІЯ

УДК 004.6

Шалаган В.С. Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 103 с.

Українською мовою. Бібліогр.: 20 назв, 12 рисунків.

У бакалаврській кваліфікаційній роботі реалізовано інформаційну систему для автоматизованого збору, обробки та збереження авіаційних даних, отриманих із агрегаторів квиткових сервісів. Основну увагу приділено побудові ETL-системи із використанням відкритих технологій, зокрема Python, Pandas, PostgreSQL та Apache Airflow. Сформовано архітектуру, що забезпечує щоденне отримання CSV-файлів через SFTP-протокол, їх обробку, трансформацію та завантаження до реляційної бази даних.

Особливу роль у системі відіграє модуль валідації: реалізовано механізм автоматичної перевірки коректності записів, який фіксує виявлені помилки у вигляді текстових коментарів. Для подальшого аналізу результатів перевірки даних розроблено компонент візуалізації — побудова кругових діаграм на Python дозволяє наочно демонструвати розподіл помилкових і валідних записів, виявляти найчастіші типи дефектів. Система є масштабованою, модульною та адаптованою до змін у форматах вхідних даних. Вона може бути інтегрована в інші корпоративні сервіси, BI-платформи або розширена за допомогою додаткових аналітичних модулів.

Ключові слова: ETL, авіаційні дані, Pandas, PostgreSQL, Apache Airflow, валідація, візуалізація даних, автоматизація, SFTP, Python.

# ABSTRACT

Shalahan V.S. Information System for Data Analysis from Airfare Aggregators Using an ETL Pipeline in Python.Bachelor's Qualification Paper, Specialty 123 "Computer Engineering", Educational Program "Computer Engineering". Vinnytsia: VNTU, 2025. 103 pages.

In Ukrainian language. Bibliography: 20 titles, 12 figures.

The bachelor's qualification presents an information system designed for automated collection, processing, and storage of airline data obtained from ticket aggregator services. The main focus is on building an ETL system using open-source technologies such as Python, Pandas, PostgreSQL, and Apache Airflow. The developed architecture enables daily retrieval of CSV files via the SFTP protocol, followed by data processing, transformation, and loading into a relational database.

A key component of the system is the validation module: it automatically checks the correctness of records and adds comments when errors are found. To support data quality control, a visualization component was created — pie charts in Python clearly show the proportion of valid and invalid records and help identify the most common types of errors.

The system is scalable, modular, and flexible to changes in input data formats. It can be integrated with other corporate services, BI platforms, or expanded with additional analytics modules.



Keywords: ETL, aviation data, Pandas, PostgreSQL, Apache Airflow, validation, data visualization, automation, SFTP, Python.

**ЗМІСТ**

# ВСТУП

**Актуальність.** Авіаційні перевезення є невід'ємною частиною глобальної транспортної системи, що забезпечує швидке сполучення між країнами, щороку кількість авіарейсів зростає, а з нею — обсяги даних, які потребують обробки, за даними IATA, у 2024 році перевезено понад 4,7 мільярда пасажирів, що створює суттєве навантаження на інформаційні системи [1].

Агрегатори квитків, як-от Trip, Kayak, Booking, Google Flights, надають доступ до тисяч пропозицій у реальному часі, однак формати даних, що надходять (CSV, JSON, XML), можуть бути неоднорідними, з дублікованими або відсутніми значеннями. Ускладнює подальший аналіз без попередньої обробки. Тому зростає потреба у побудові ефективного ETL-процесу (Extract — витягнення, Transform — перетворення, Load — завантаження), що забезпечує автоматизоване вилучення, очищення та збереження даних у придатному для аналізу форматі [2,3].

Сучасні наукові роботи активно досліджують способи автоматизації обробки даних у сфері транспорту з використанням відкритих технологій — Pandas, PostgreSQL, Apache Airflow. Увага приділяється як структуризації, так і перевірці якості вхідних даних [4,5].

Метою бакалаврської кваліфікаційної роботи є розробка інформаційної системи, здатної автоматизовано отримувати дані з агрегаторів авіаперевезень, обробляти їх відповідно до вимог до якості, зберігати у централізованому сховищі та готувати до подальшого аналізу.

Для досягнення мети передбачено виконання таких завдань:

— проаналізувати особливості даних, що надходять з агрегаторів авіаперевезень, та визначити вимоги до їх обробки;

— вивчити сучасні технології для реалізації ETL-процесів, зокрема Python, Pandas, PostgreSQL та Apache Airflow;

— розробити архітектуру інформаційної системи для збору, трансформації та збереження даних;

— реалізувати ETL Pipeline (конвеєр) для автоматизованого завантаження даних з SFTP-серверів у форматі CSV;

— здійснити валідацію даних на етапі обробки;

— налаштувати механізми автоматичного запуску та моніторингу процесів за допомогою Apache Airflow;

— реалізувати систему візуалізації результатів валідації даних та провести її апробацію в контексті контролю якості;

— провести тестування системи та оцінити її ефективність і надійність у роботі з реальними даними.

**Об’єкт дослідження:** процес побудови ETL-процесів для автоматизованого збору, обробки та аналізу даних з агрегаторів авіаперевезень.

**Предмет дослідження:** методи, інструменти та технології реалізації ETL Pipeline із застосуванням Python, Pandas, PostgreSQL та Apache Airflow для створення інформаційної системи аналізу авіаційних даних.

**Методи дослідження:** у роботі застосовано методи аналізу структурованих даних, проєктування програмного забезпечення, автоматизації ETL-процесів, програмної реалізації з використанням відкритого ПЗ, а також методи візуалізації статистичних результатів і валідації даних на основі Python-бібліотек.

**Наукова новизна одержаних результатів:** розроблено систему, що забезпечує обробку неоднорідних авіаційних даних з валідацією та візуалізацією типових помилок, реалізовано масштабований ETL-процес з відкритими інструментами, що може бути адаптований до інших форматів або джерел.

**Апробація результатів:** матеріали кваліфікаційної роботи були подані у вигляді тез доповіді «ETL PIPELINE НА PYTHON ДЛЯ АНАЛІЗУ ДАНИХ АГРЕГАТОРІВ У СФЕРІ АВІАПЕРЕВЕЗЕНЬ» на LIV Всеукраїнську науково-технічну конференцію факультету інформаційних технологій та комп’ютерної інженерії (2025).

# 1 ОГЛЯД ТЕХНОЛОГІЙ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ETL-СИСТЕМИ

## 1.1 Поняття та принципи ETL-системи

ETL означає Extract, Transform, Load — процес, який використовується в сховищах даних для вилучення даних із різних джерел, перетворення їх у формат, придатний для завантаження в сховище даних, а потім завантаження в сховище.

Процес ETL — ітераційний процес, який повторюється, коли нові дані додаються до сховища. Цей процес важливий, оскільки він гарантує, що дані в сховищі даних є точними, повними та актуальними. Він допомагає переконатися, що дані мають формат, необхідний для аналізу даних і звітності.

Інструмент ETL витягує дані з різних систем джерел даних, перетворює їх у проміжну область, а потім, нарешті, завантажує в систему сховища даних (рис 1.1).

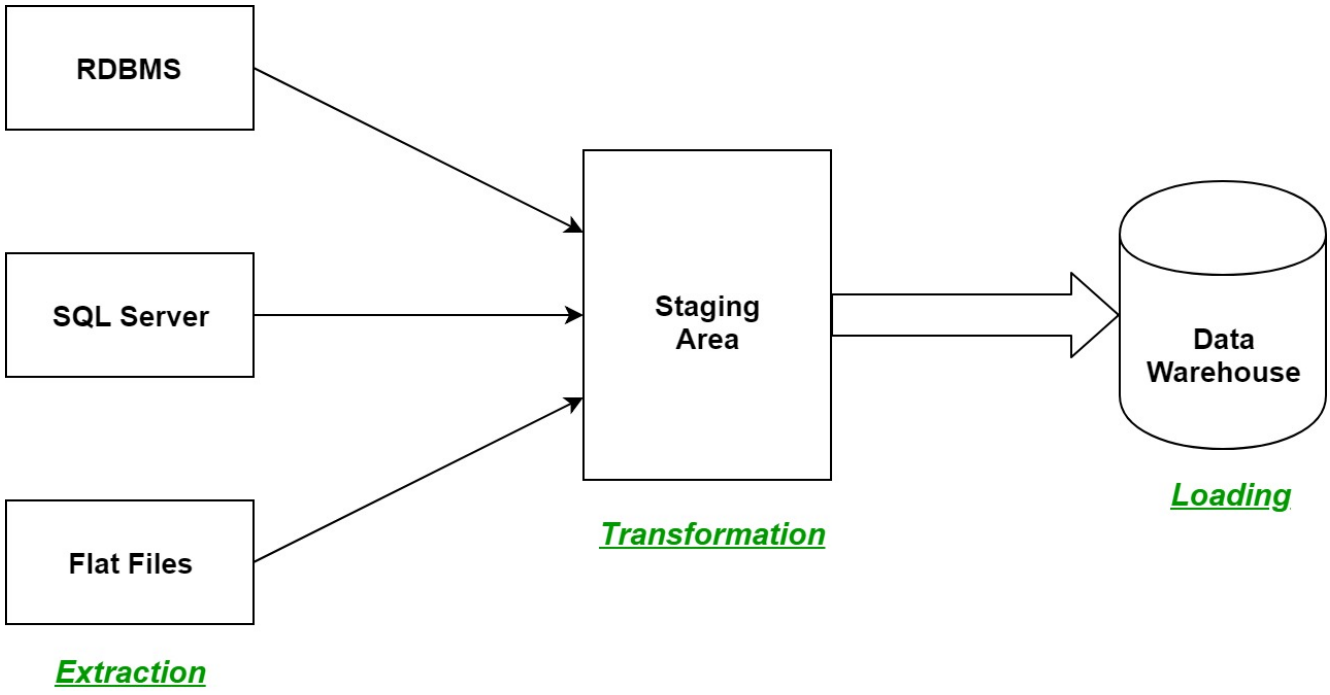



Рисунок 1.1 — Блок – схема роботи ETL

Першим кроком процесу ETL є витягнення. На цьому кроці витягуються дані з різних вихідних систем, які можуть бути в різних форматах, таких як реляційні бази даних, без SQL, XML і плоских файлів у проміжну область. Важливо витягувати дані з різних вихідних систем і зберігати їх спочатку в проміжній

області, а не безпосередньо в сховищі даних, оскільки витягнуті дані мають різні формати та можуть бути пошкоджені. Тому завантаження безпосередньо в сховище даних може пошкодити його, і відновлення буде набагато складнішим. Тому це один із найважливіших етапів процесу ETL.

Другим кроком процесу ETL є перетворення. На цьому етапі набір правил або функцій застосовуються до витягнутих даних, щоб перетворити їх у єдиний стандартний формат.

Включає такі процеси/завдання:

— фільтрування — завантаження в сховище даних лише певних атрибутів;

— очищення — заповнення значень NULL деякими значеннями за замовчування;.

— об’єднання — об’єднання кількох атрибутів в один;

— розбиття — розбиття одного атрибута на кілька атрибутів.

Третім і останнім кроком процесу ETL є завантаження. На цьому етапі перетворені дані остаточно завантажуються в сховище даних. Іноді дані оновлюються шляхом дуже частого завантаження в сховище даних, а іноді це робиться через триваліші, але регулярні проміжки часу. Швидкість і період завантаження залежать виключно від вимог і відрізняються від системи до системи.

Процес ETL може використовувати концепцію конвеєрної обробки, тобто, як тільки деякі дані витягуються, їх можна трансформувати, і протягом цього періоду деякі нові дані можуть бути вилучені. І поки трансформовані дані завантажуються в сховище даних, уже витягнуті дані можна трансформувати (рис 1.2).

Переваги процесу ETL у сховищах даних:

— покращена якість даних: процес ETL гарантує, що дані в сховищі даних є точними, повними та актуальними;

— краща інтеграція даних: процес ETL допомагає інтегрувати дані з багатьох джерел і систем, роблячи їх більш доступними та зручними для використання;

— підвищена безпека даних: процес ETL може допомогти покращити безпеку даних, контролюючи доступ до сховища даних і забезпечуючи доступ до даних лише авторизованим користувачам;

— покращена масштабованість: процес ETL може допомогти покращити масштабованість, надаючи спосіб керування та аналізу великих обсягів даних;

— підвищена автоматизація: інструменти та технології ETL можуть автоматизувати та спростити процес ETL, зменшуючи час і зусилля, необхідні для завантаження та оновлення даних у сховищі.

Недоліки процесу ETL у сховищах даних:

— висока вартість: процес ETL може бути дорогим для впровадження та підтримки, особливо для організацій з обмеженими ресурсами;

— складність : процес ETL може бути складним і складним у реалізації, особливо для організацій, яким бракує необхідного досвіду чи ресурсів;

— обмежена гнучкість: процес ETL може бути обмежений з точки зору гнучкості, оскільки він може не мати змоги обробляти неструктуровані дані або потоки даних у реальному часі;

— обмежена масштабованість : процес ETL може бути обмежений з точки зору масштабованості, оскільки він не зможе обробляти дуже великі обсяги даних;

— занепокоєння конфіденційністю даних : процес ETL може викликати занепокоєння щодо конфіденційності даних, оскільки збираються, зберігаються та аналізуються великі обсяги даних.

Загалом процес ETL є важливим процесом у сховищах даних, який допомагає гарантувати, що дані в сховищі даних є точними, повними та актуальними. Однак це також пов’язано зі своїми труднощами та обмеженнями, і організації повинні ретельно проаналізувати витрати та вигоди перед їх впровадженням [6].

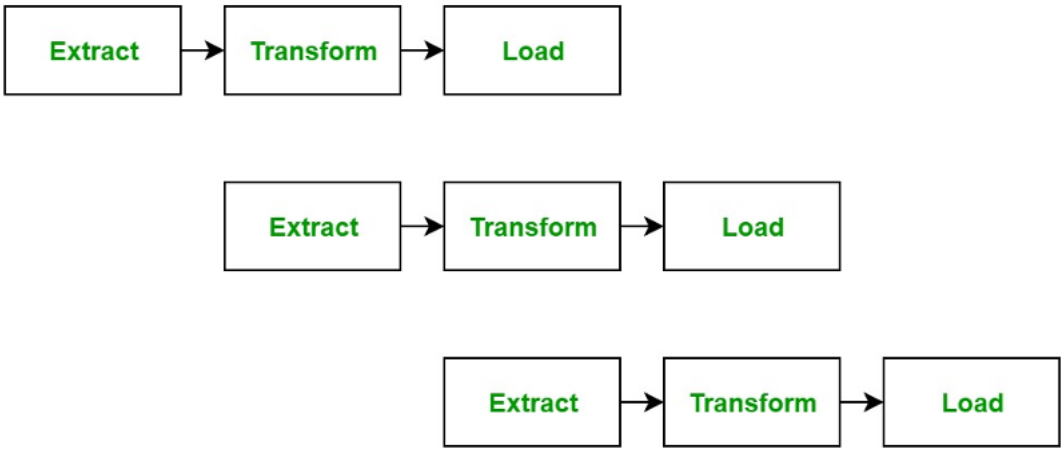



Рисунок 1.2 — Блок-схема конвеєрного процесу ETL

## 1.2 Специфікація структури даних

Розробка ефективного ETL-процесу неможлива без чітко структурованого опису даних, з якими працює система. Саме тому створення повної специфікації структури таблиці стало важливим етапом у побудові інформаційної системи. Специфікація визначає: які поля має містити кожен запис, які з них є обов'язковими, які типи даних використовуються, за якою логікою перевіряється їх відповідність тощо. Цей підхід дозволяє уніфікувати обробку даних та мінімізувати помилки при завантаженні з CSV-файлів у систему [7].

Основою вхідних даних є CSV-файли, що надходять з SFTP-серверів. У таких файлах може міститись інформація про ціни, маршрути, перевізників, час вильоту/прильоту тощо. Для кожного поля було визначено тип та правила обов'язковості — залежно від контексту. Поле check_out_date (дата виїзду) є обов'язковим лише у випадку, якщо рейс є зворотним is_one_way = No (односторонній рух = Ні). Таким чином, структура дозволяє адаптуватись до особливостей кожного запиту, не втрачаючи логіки та цілісності даних.

Формати дати та часу в специфікації відповідають міжнародному стандарту ISO 8601: YYYY-MM-DD для дати і HH:MM:SS або HH:MM для часу.

ISO 8601 позначає дату та час, починаючи з року, за яким слідують місяць, день, година, хвилини, секунди та мілісекунди. Що дозволяє уникнути неоднозначностей при зберіганні даних, забезпечити сумісність з більшістю сучасних СУБД (зокрема, PostgreSQL) та бібліотеками Python, як-от datetime (дата й час), pandas.to_datetime() тощо [8].

Критично важливою є підтримка авіаційних кодів та міжнародних стандартів. Зокрема, у полях origin (відправлення) та destination (призначення) використовуються IATA-коди (LHR для Лондона, JFK для Нью-Йорка). Такий підхід забезпечує уніфікацію із зовнішніми системами (агрегаторами, API), які оперують цими стандартами. Полегшує подальшу інтеграцію, порівняння пропозицій, агрегацію й аналіз [9].

Поле currency (валюта) стандартизовано відповідно до ISO 4217, де використовуються трилітерні коди валют (USD, UAH, EUR).

ISO 4217 — стандарт, опублікований Міжнародною організацією зі стандартизації (ISO), який визначає алфавітні та цифрові коди для представлення валют і надає інформацію про зв’язки між окремими валютами та їхніми меншими одиницями [10].

Поля, пов’язані з зупинками (stopovers), мають додаткову логіку перевірки: якщо outbound_stops (вихідні зупинки) або inbound_stops (вхідні зупинки) > 0, тоді відповідно мають бути заповнені *_travel_stop_over (кількість зупинок в подорожі) та *_travel_stop_over_duration (тривалість зупинок). Дозволяє точно вказати маршрут у разі пересадок.

У частині фінансових даних поля: price_inc (ціна з урахуванням), price_exc (ціна без урахування), price_outbound (ціна на вихідні дані), price_inbound (ціна вхідна), tax (податки), передбачено два варіанти: загальна вартість з податками або без них. Важливо, оскільки різні агрегатори надають дані у різних форматах. Kayak і Google Flights можуть передавати тільки кінцеву вартість, тоді як внутрішні API окремо виділяють податкову складову. Також було передбачено поле is_tax_inc_outin (вихідний розрахунковий податок), яке фіксує, чи входять податки до ціни (1) чи ні (0).

Окремі поля: Coupon_code (код купону), outbound_seats (кількість місць), inbound_fare_family (сімейний вихідний тариф) наразі не використовуються, але збережені як зарезервовані для майбутнього функціонального розширення. Наявність таких полів дозволяє уникати повторного моделювання структури таблиці у майбутньому, що відповідає принципам forward-compatibility (сумісність уперед) у розробці інформаційних систем.

Таким чином, розроблена структура поєднує в собі суворість перевірки, дотримання міжнародних стандартів, підтримку динамічних умов та готовність до розширення. Вона є основою для надійного функціонування всієї ETL-системи та гарантує якість і цілісність даних у процесі обробки.

## 1.3 Особливості аналізу даних агрегаторів у сфері авіаперевезень

### 1.3.1 Поняття туристичні агрегатори

Туристичні агрегатори — онлайн-платформи або вебсайти, які збирають інформацію від різних туристичних постачальників, таких як авіакомпанії, готелі та компанії з прокату автомобілів, і представляють її користувачам у консолідованому форматі. Ці агрегатори дозволяють мандрівникам порівнювати ціни, бронювати авіаквитки, житло та інші туристичні послуги в одному місці.

Туристичні агрегатори стали важливими інструментами для мандрівників у сучасну епоху. Вони спрощують процес планування та бронювання поїздок, пропонуючи зручність, економію коштів та широкий вибір можливостей. Із розвитком Інтернету та онлайн бронювання подорожей агрегатори стали основними ресурсами для мандрівників, які шукають найкращі пропозиції та варіанти.

Агрегатори авіаперевезень відіграють ключову роль у сучасній туристичній індустрії, допомагаючи мандрівникам швидко та ефективно знаходити оптимальні варіанти перельотів. Основною особливістю цих платформ є обробка великих обсягів даних у реальному часі, що вимагає застосування сучасних алгоритмів аналізу та прогнозування. Важливими аспектами такої роботи є збір, обробка, систематизація та аналіз інформації з різних джерел.

### 1.3.2 Особливості аналізу даних

Дані, які використовуються агрегаторами, надходять від авіакомпаній, глобальних дистрибутивних систем (GDS), аеропортів та інших постачальників. API авіакомпаній дозволяють отримувати інформацію про наявність місць, зміни в розкладі рейсів та актуальні тарифи. Глобальні дистрибутивні системи, такі як Amadeus, Sabre та Travelport, консолідують дані від авіаперевізників і дозволяють агрегаторам формувати єдину базу для порівняння варіантів перельоту. Крім того, важливими джерелами є аеропорти, які передають дані про можливі затримки, погодні умови та зміни в маршрутах.

Оскільки ціни на авіаквитки змінюються динамічно, особливу увагу приділяють аналізу цінових тенденцій. Для цього використовують методи машинного навчання та статистичні алгоритми, що дозволяють прогнозувати майбутні зміни тарифів. Враховується безліч факторів, таких як сезонність, рівень попиту, економічні умови та стратегія авіакомпаній щодо ціноутворення. Наприклад, у пікові періоди святкових подорожей ціни можуть змінюватися кілька разів на день, а алгоритми агрегаторів повинні вчасно реагувати на такі зміни, щоб надавати користувачам актуальну інформацію.

Ще однією особливістю є оптимізація пошуку та ранжування результатів. Використовуються алгоритми побудови маршрутів, що дозволяють знаходити найкращі варіанти перельоту з урахуванням ціни, тривалості, кількості пересадок та інших факторів. Якщо користувач шукає найдешевший варіант, система може запропонувати рейси з довгими стикуваннями, тоді як для тих, хто хоче зекономити час, будуть показані прямі перельоти або варіанти з мінімальними пересадками.

Персоналізація відіграє важливу роль у покращенні користувацького досвіду. Агрегатори аналізують історію пошуку, вподобання та попередні бронювання, щоб надавати більш релевантні рекомендації. Використовуючи штучний інтелект, система може пропонувати найкращі варіанти для конкретного користувача, якщо він зазвичай обирає певні авіакомпанії або уникає рейсів з довгими пересадками.

Однак аналіз даних агрегаторів авіаперевезень стикається з низкою викликів. По-перше, це висока динамічність тарифів, що вимагає постійного оновлення

інформації в реальному часі. По-друге, обробка великих обсягів даних вимагає застосування сучасних технологій, таких як Hadoop, Spark та хмарні обчислення, що дозволяють швидко аналізувати мільйони запитів щодня. По-третє, конфлікти даних з різних джерел можуть спричиняти розбіжності у відображенні цін, що змушує агрегатори розробляти спеціальні механізми перевірки достовірності інформації.

Крім того, важливими є питання регулювання та дотримання правових норм, особливо щодо обробки персональних даних. Європейський регламент (GDPR) та інші подібні акти вимагають від агрегаторів забезпечення безпеки користувацької інформації, а також прозорості у відображенні тарифів та умов бронювання.

Для подальшого покращення аналітики агрегатори впроваджують передові технології, такі як штучний інтелект, нейромережі та блокчейн. Використання AI дозволяє автоматично виявляти шахрайські схеми, наприклад, підозріло великі зміни цін або маніпуляції з рейтингами. Графові бази даних допомагають знаходити оптимальні маршрути, враховуючи складні залежності між рейсами, а блокчейн може забезпечити прозорість транзакцій та підвищити довіру до агрегаторів.

Загалом, аналіз даних у сфері авіаперевезень є складним і багатогранним процесом, що потребує комплексного підходу. Використання сучасних технологій, ефективне прогнозування та персоналізація дозволяють агрегаторам не лише надавати користувачам найкращі пропозиції, а й залишатися конкурентоспроможними у динамічному середовищі туристичних послуг.

## 1.4 Порівняльний аналіз інструментів для реалізації ETL

### 1.4.1 Порівняння Dask  DataFrames і Pandas DataFrames

Pandas широко використовувана бібліотека Python для обробки та аналізу даних. Він надає об’єкт DataFrame, який є двовимірною неоднорідною табличною структурою даних із змінним розміром.

Плюси Pandas DataFrames:

— простота використання: Pandas відомий своїм зручним та інтуїтивно зрозумілим API, що робить його популярним вибором для аналітиків даних і вчених, які починають працювати з даними;

— багата функціональність: Pandas пропонує широкий набір функцій для очищення, трансформації, групування, агрегації та візуалізації даних, його повна документація та активне співтовариство дозволяють відносно легко знаходити рішення типових завдань;

— обробка в пам’яті: Pandas працює виключно в пам’яті, що може призвести до швидкого виконання для наборів даних, які вміщуються в межах доступної пам’яті.

Мінуси Pandas DataFrames:

— обмеження пам’яті: одним із головних недоліків Pandas є обмеження пам’яті, він може важко обробляти набори даних, які занадто великі, щоб помістится в пам’ять, що призводить до вузьких місць продуктивності або збоїв;

— обмежений паралелізм: хоча Pandas пропонує певну підтримку паралелізму, він може не повністю використовувати всі доступні ядра ЦП для складних операцій.

Dask — бібліотека з відкритим вихідним кодом, розроблена для обробки наборів даних, які перевищують обсяг пам’яті, і паралельних обчислень. Dask DataFrames розширює API Pandas, щоб забезпечити паралельні та розподілені обчислення в різних обчислювальних середовищах.

Плюси Dask DataFrames:

— масштабованість: Dask DataFrames може обробляти набори даних, розмір яких перевищує доступну пам’ять, розбиваючи їх на менші розділи, які можна обробляти паралельно, забезпечує ефективну обробку великих даних без обмежень пам’яті;

— паралельні та розподілені обчислення: Dask за своєю суттю підтримує паралелізм, дозволяючи виконувати операції на кількох ядрах ЦП або навіть розподіляти між кластером машин, що призводить до значного підвищення продуктивності для інтенсивних обчислювальних завдань;

— ліниве оцінювання: стратегія ледачого оцінювання Dask оптимізує обчислення шляхом побудови обчислювального графіка операцій, такий підхід зменшує витрати пам’яті та підвищує загальну ефективність.

Мінуси Dask DataFrames:

— крива навчання: хоча API Dask схожий на Pandas, він має деякі відмінності, користувачам, які знайомі з Pandas, можливо, доведеться витратити деякий час на розуміння паралельних і розподілених концепцій Dask;

— складність: розподілена природа Dask може внести складність, особливо при роботі з кластерами та розподіленими середовищами, налаштування та налаштування кластера Dask може потребувати додаткових зусиль.

Використовуйте Pandas DataFrames, коли:

— набір даних зручно розміщується в доступній пам’яті;

— віддаєте пріоритет простоті використання та швидкому навчанню;

— потрібно виконувати завдання з обробки даних на одній машині;

— паралелізм не є критичною вимогою для ваших завдань.

Використовуйте Dask DataFrames, коли:

— набір даних завеликий, щоб помістится в пам’ять;

— потрібен паралелізм для швидшого обчислення завдань обробки даних;

— хочете скористатися перевагами розподіленого обчислення на багатоядерній машині або в кластері;

— готові витратити час на вивчення концепцій Dask заради масштабованості та ефективності.

І Pandas DataFrames, і Dask DataFrames мають свої сильні та слабкі сторони, що робить їх придатними для різних сценаріїв. Якщо маєте справу з відносно невеликими наборами даних і надаєте перевагу простоті, Pandas може бути правильним вибором. З іншого боку, якщо працюєте з великими даними та потребуєте ефективних паралельних або розподілених обчислень, Dask DataFrames пропонує вражаюче рішення. Вибір, зрештою, залежить від розміру даних, складності завдань і вашої готовності досліджувати та застосовувати нові інструменти для оптимальної продуктивності та масштабованості [11].

### 1.4.2 Airflow проти Luigi: ключові відмінності

Оркестрація процесів обробки даних є невіддільною частиною сучасної інженерії даних. Вона забезпечує автоматичне керування ETL-системою, контроль залежностей між етапами та підвищення надійності виконання. Серед найбільш поширених інструментів — Apache Airflow і Luigi. Обидва орієнтовані на Python, підтримують автоматизацію завдань і мають відкритий вихідний код, але різняться за архітектурою, функціональністю та масштабованістю.

Apache Airflow — потужна платформа для створення, планування та моніторингу складних робочих процесів. Вона ґрунтується на DAG-моделі (орієнтований ациклічний граф), де кожне завдання має чітко визначені залежності. Завдяки такій структурі забезпечується контроль за послідовністю виконання задач, а також можливість паралельної обробки.

Luigi, розроблений Spotify, базується на описі задач у вигляді класів Python з прямим зазначенням залежностей. Замість попереднього формування графа, дерево задач формується динамічно під час виконання. Такий підхід дозволяє швидко реалізовувати лінійні або умовно послідовні робочі процеси без додаткових конфігурацій.

Airflow підтримує розподілене виконання задач за допомогою виконавців: LocalExecutor, CeleryExecutor, KubernetesExecutor. Дає змогу ефективно масштабувати обчислення у великих системах. Архітектура дозволяє використовувати інтеграції з хмарними платформами, API, базами даних і системами повідомлень.

Luigi реалізує більш легку архітектуру з одним центральним планувальником. Виконання задач відбувається послідовно або з обмеженим паралелізмом. Відсутність підтримки кластерного виконання обмежує використання у високонавантажених середовищах, однак забезпечує простоту розгортання та мінімальні вимоги до інфраструктури.

Планування задач у Airflow є гнучким і деталізованим. Користувач задає розклад виконання, умови повторного запуску, таймаути, залежності від зовнішніх

подій або файлів. Планувальник підтримує запуск задач у визначений час або за зовнішніми тригерами, що дозволяє створювати динамічні та реактивні системи.

У Luigi запуск задач здійснюється через CLI (інтерфейс командного рядка) або cron. Система планування обмежена базовими можливостями, без глибокої підтримки повторних спроб чи динамічних умов виконання. У разі потреби складної логіки запуску доводиться реалізовувати її вручну у коді.

Airflow має зручний вебінтерфейс, де відображаються всі DAG, статуси задач, лог-файли, тривалість виконання та історія запусків. Інтерфейс дозволяє зручно перезапускати задачі, аналізувати помилки та контролювати продуктивність пайплайнів.

Luigi не має повноцінного вебінтерфейсу. Візуалізація задач обмежена спрощеною сторінкою або логами у терміналі. Ускладнює моніторинг у великих проєктах, але залишається достатнім у невеликих сценаріях з невеликою кількістю завдань.

Airflow має велику кількість вбудованих інтеграцій — з AWS, GCP, Azure, FTP, HTTP, SMTP, Slack, Docker, базами даних та іншими сервісами. Завдяки модульній структурі легко додаються власні оператори та розширення.

Luigi підтримує базову інтеграцію з файловими системами, Hadoop, SQL, але загальна кількість готових конекторів менша. Розширення можливе за рахунок написання кастомних задач, однак це вимагає більше ручної роботи та глибшого занурення в код.

У складних корпоративних середовищах, де необхідне масштабування, контроль стану задач і гнучкість запуску, Airflow демонструє високу ефективність. Його обирають у системах із великою кількістю залежностей, необхідністю взаємодії з зовнішніми API та високими вимогами до надійності.

Luigi доцільно використовувати у проєктах середнього або малого масштабу. Він підходить для автоматизації щоденних ETL-процесів, де структура проста, а складні сценарії запуску не є критичними. Завдяки легкості у використанні він особливо зручний для невеликих команд або одиночних розробників.

І Airflow, і Luigi вимагають знання Python. Для роботи з Airflow потрібні базові навички DevOps, розуміння мережевої архітектури та досвід роботи з хмарними рішеннями. Luigi менш вимогливий до технічного бекграунду, що робить його більш доступним у простих сценаріях.

Airflow забезпечує високу масштабованість, підтримує десятки тисяч задач одночасно та дозволяє керувати процесами в реальному часі. Однак така потужність потребує відповідної інфраструктури, налаштування баз даних, брокерів черг, систем зберігання логів.

Luigi вимагає менше ресурсів і простіший у розгортанні. Його можна швидко запустити локально без складних конфігурацій. Знижує витрати часу на підтримку, але обмежує в розширенні функціональності при зростанні проєкту.

Обираючи між Airflow і Luigi, необхідно враховувати складність робочих процесів, обсяг даних, потреби в масштабуванні та технічні можливості команди. Airflow підходить для великого бізнесу, довготривалих розгортань і складних сценаріїв. Luigi — для локальних задач, прототипування та простих автоматизацій.

У довгостроковій перспективі обидва інструменти залишаються актуальними. Airflow займає позицію стандарту для складних систем оркестрації. Luigi залишається надійним вибором для швидкого старту та невеликих ETL-завдань з мінімальними залежностями [12].

## 1.5 Порівняльний аналіз засобів візуалізації: matplotlib.pyplot vs Tableau

Tableau та Matplotlib.pyplot — два потужні інструменти у світі аналізу даних, але вони служать різним цілям. Tableau відомий своєю простотою у використанні та можливістю швидкого створення візуалізацій. Інтерфейс Tableau дозволяє користувачам створювати графіки без необхідності писати код, що робить його доступним навіть для нетехнічних спеціалістів. Matplotlib.pyplot, з іншого боку, є бібліотекою для створення візуалізацій, що дозволяє отримати повний контроль над процесом через програмування. Він пропонує більшу гнучкість у налаштуваннях, дозволяючи створювати складніші та індивідуалізовані графіки.

Порівнюючи ці два інструменти, спеціалісти з обробки даних можуть вибрати той, який найкраще відповідає конкретним вимогам. Якщо потрібно швидко створювати інтерактивні панелі чи стандартні графіки, Tableau буде ідеальним вибором. Якщо ж необхідна висока гнучкість, кастомізовані графіки або складні маніпуляції з даними, Matplotlib.pyplot стане кращим інструментом.

Незалежно від того, чи зосереджені на створенні простих візуалізацій, чи потрібні розширеніші можливості обробки даних, розуміння сильних сторін кожного інструменту допоможе зробити правильний вибір для конкретних проектів.

### 1.5.1 Огляд Tableau

Tableau — потужний програмний інструмент для візуалізації та аналітики даних, який активно використовується в бізнесі, наукових дослідженнях і сфері прийняття управлінських рішень. Основна мета Tableau полягає в наданні можливості користувачам швидко й ефективно інтерпретувати великі обсяги інформації за допомогою інтерактивних графіків, діаграм і інформаційних панелей. Його функціонал орієнтований як на аналітиків, так і на користувачів без технічної підготовки — завдяки інтуїтивно зрозумілому інтерфейсу та функції перетягування елементів.

Tableau підтримує підключення до широкого спектра джерел даних — від локальних таблиць до хмарних баз даних — і дозволяє створювати візуалізації без написання коду. Забезпечує гнучкість у роботі з різними форматами даних та прискорює процес аналізу. Завдяки широкому набору шаблонів, розширеній бібліотеці графіків і наявності інтерактивних компонентів, Tableau ефективно використовується для побудови інформаційних панелей у реальному часі.

Основні функціональні можливості Tableau:

— інтерфейс перетягування елементів (drag-and-drop), що спрощує побудову звітів;

— підтримка аналізу даних у режимі реального часу;

— інтеграція з численними джерелами даних — як локальними, так і віддаленими;

— використання попередньо створених шаблонів та панелей;

— широкий набір засобів для побудови візуалізацій;

— наявність інструментів безпеки для захисту чутливих даних;

— можливості спільної роботи та обміну результатами через веб або хмару.

Переваги використання Tableau:

— зручність для користувачів без програмістського досвіду;

— висока швидкість створення наочних візуалізацій;

— активна спільнота користувачів і доступ до великої кількості навчальних матеріалів;

— регулярне оновлення функціоналу та впровадження нових можливостей;

— ефективність для задач бізнес-аналітики та щоденного моніторингу показників;

— підтримка інтерактивних і динамічних панелей управління;

— здатність працювати з великими наборами даних без значної втрати продуктивності.

Основні обмеження Tableau:

— менші можливості налаштування в порівнянні з програмними мовами;

— висока вартість корпоративного ліцензування;

— складність засвоєння деяких розширених функцій;

— потреба в потужній інфраструктурі даних при роботі з великими обсягами;

— обмеженість інструментів для попередньої обробки та трансформації даних;

— недостатня підтримка складної статистики та машинного навчання;

— менша гнучкість у створенні сценаріїв, ніж у Python або R;

— залежність від стабільності зовнішніх джерел даних;

— технічна складність інтеграції з деякими сторонніми платформами.

Tableau залишається одним із найпопулярніших рішень для візуалізації даних, завдяки поєднанню гнучкості, функціональності та зручності у використанні. Незважаючи на певні обмеження, він є ефективним інструментом для підтримки прийняття рішень на основі даних у різних галузях.

### 1.5.2 Огляд matplotlib.pyplot

Matplotlib.pyplot у Python служить основним інструментом для візуалізації даних, що дозволяє користувачам створювати широкий спектр графіків і діаграм, від гістограм до точкових діаграм. Ця бібліотека полегшує перетворення даних у візуальний контекст, полегшуючи розуміння складної інформації. За допомогою Matplotlib.pyplot користувачі можуть налаштовувати графіки відповідно до своїх потреб, регулюючи кольори, розміри та типи графіків. Бібліотека підтримує різні серверні модулі, підвищуючи її універсальність на різних платформах. Інтеграція з Pandas і NumPy оптимізує процес обробки даних, забезпечуючи ефективний аналіз. Інтерактивні функції в Matplotlib.pyplot покращують взаємодію користувачів з даними. Можливості експорту забезпечують спільний доступ до візуалізацій у кількох форматах. Велика документація бібліотеки та підтримка спільноти роблять її доступною як для початківців, так і для професіоналів. Розширені функції, такі як тривимірне малювання та анімація, відкривають додаткові можливості для динамічного представлення даних. Послідовні оновлення Matplotlib.pyplot зберігають його актуальність і потужність для сучасних завдань візуалізації даних [13].

Переваги використання бібліотеки Matplotlib.pyplot:

— широкі можливості візуалізації даних;

— простота використання для базових графіків;

— гнучкість і налаштування;

— інтеграція з іншими бібліотеками;

— підтримка 3D графіків;

— розробка з відкритим кодом.

Недоліки використання Matplotlib.pyplot:

— крута крива навчання для складних графіків;

— обмежена інтерактивність;

— погана продуктивність з великими наборами даних;

— стандартні графіки не завжди естетично привабливі.

Обмеження бібліотеки Matplotlib.pyplot:

— обмежена підтримка 3D-графіків;

— важкість у створенні анімованих графіків;

— менша підтримка інтерактивних елементів.

### 1.5.3 Tableau проти Matplotlib.pyplot

При виборі правильного інструменту для візуалізації даних часто порівнюють Tableau і Matplotlib.pyplot. Обидва інструменти мають свої сильні сторони і можуть бути потужними в різних ситуаціях. Розуміння відмінностей між ними допомагає визначити, який з них найкраще відповідає потребам.

Tableau відомий зручним інтерфейсом. Можна швидко створювати візуалізації за допомогою функцій перетягування. Інструмент розроблений для користувачів, які не мають достатнього технічного досвіду. Matplotlib.pyplot, з іншого боку, є потужною бібліотекою для візуалізації даних, яка вимагає певних знань програмування. Вона пропонує гнучкість і контроль на всіх етапах створення графіків, але має круту криву навчання для новачків.

Matplotlib.pyplot відрізняється гнучкістю та можливістю налаштування. Повний контроль надається над усіма аспектами графіків: від вибору типів графіків до налаштувань елементів (ліній, кольорів, маркерів тощо). Tableau має обмежену гнучкість порівняно з Matplotlib.pyplot і підходить для стандартних візуалізацій, хоча для дуже індивідуалізованих або складних графіків може знадобитися обхідні шляхи.

Що стосується обробки даних, Tableau виділяється здатністю швидко підключатися до різних джерел даних і обробляти великі набори даних без особливих зусиль. Дозволяє швидко отримувати результати. Matplotlib.pyplot

вимагає більше часу на налаштування, але надає більший контроль для складніших візуалізацій.

Tableau має зручний інтерфейс, що робить його доступним для тих, хто не має досвіду кодування. Графіки і панелі можна створювати за кілька кліків миші. Matplotlib.pyplot вимагає знань у програмуванні, тому освоєння бібліотеки може зайняти більше часу, але в результаті відкривається великий потенціал для індивідуалізації візуалізацій.

Tableau легко інтегрується з популярними бізнес-інструментами та базами даних, що спрощує включення в існуючі робочі процеси. Він добре працює з такими інструментами, як Salesforce, Google Analytics і Excel. Matplotlib.pyplot інтегрується з іншими інструментами, але для цього необхідно виконати налаштування через код, що вимагає знань у програмуванні.

Tableau є платним програмним забезпеченням з різними ціновими рівнями, в залежності від функцій, які необхідні. Вартість може бути високою, особливо для малого бізнесу. Matplotlib.pyplot є безкоштовною бібліотекою з відкритим кодом, що робить її економічно ефективним вибором.

Tableau має сильну спільноту і офіційну підтримку, що полегшує пошук допомоги та ресурсів. Matplotlib.pyplot має велику активну спільноту, з численними форумами і документацією, доступними в Інтернеті, хоча підтримка здебільшого здійснюється через спільноту.

Tableau краще підходить для:

— швидка інформація про бізнес;
— інтерактивні інформаційні панелі;
— нетехнічні користувачі;
— попередньо створені інтеграції з бізнес-інструментами;

Matplotlib.pyplot краще підходить для:

— комплексний аналіз даних;
— індивідуальні візуалізації;
— створення кастомізованих візуалізацій за допомогою коду;
— розширене маніпулювання даними та машинне навчання [14].

# 2 РЕАЛІЗАЦІЯ ETL-СИСТЕМИ АГРЕГАТОРА АВІАПЕРЕВЕЗЕНЬ

## 2.1 Опис вхідних даних та вибір методів їх обробки

### 2.1.1 Завантаження вхідних файлів з SFTP-сервера

Завантаження файлів з SFTP-сервера на локальний комп'ютер. Для цього використовуються бібліотеки Python, зокрема paramiko, яка дозволяє здійснювати підключення до SFTP-сервера та здійснювати передачу файлів.

Перед тим як розпочати завантаження файлів, необхідно налаштувати параметри підключення до SFTP-сервера. Що включає хост, порт, ім'я користувача та пароль для аутентифікації. Параметри виглядають наступним чином:

— `sftp_host` — хост (IP-адреса або доменне ім'я сервера);

— `sftp_port` — стандартний порт для SFTP — 22;

— `sftp_username` та `sftp_password` — облікові дані для аутентифікації;

— `remote_directory` — директорія на сервері, з якої будуть завантажуватися файли (лістинг 2.1).

Лістинг 2.1 — Завантаження файлів з SFTP-сервера

```
sftp_host = "my_host"
sftp_port = 22
sftp_username = "my_user"
sftp_password = "my_password"
remote_directory = "my_remote_directory"
```

Наступним кроком є встановлення максимального розміру файлів для завантаження. Обмежити завантаження лише тими файлами, що мають розмір до 100 МБ**.** Це дозволяє зменшити навантаження на систему під час обробки великих обсягів даних та уникнути потенційних збоїв у роботі ETL-процесу. Крім того, обмеження сприяє покращенню продуктивності та швидшій обробці вхідної інформації.

`MAX_FILE_SIZE_MB` — максимальний розмір файлу в мегабайтах.

`MAX_FILE_SIZE_BYTES` — максимальний розмір файлу, переведений у байти для точного порівняння (лістинг 2.2).

Лістинг 2.2 — Перевірка розміру файлів для завантаження

```
MAX_FILE_SIZE_MB = 100
MAX_FILE_SIZE_BYTES = MAX_FILE_SIZE_MB * 1024 * 1024
```

Завдяки параметрам підключення і розміру файлу можна створити функцію для завантаження файлів. Якщо розмір файлу менший за максимальний ліміт, то він буде завантажений на локальний диск.

`os.makedirs(local_directory, exist_ok=True)` — створює локальну директорію для збереження файлів, якщо вона не існує (лістинг 2.3).

Лістинг 2.3 — Функція для завантаження файлів

```
    def downloader(current_date: str):
        local_directory = os.path.join(os.path.expanduser("~"), 
"downloads", current_date)
        os.makedirs(local_directory, exist_ok=True)
        print(f"📂 Локальна директорія для 
завантаження: {os.path.abspath(local_directory)}")
```

Функція download_file призначена для завантаження файлів з віддаленого SFTP-сервера на локальний диск. Вона перевіряє розмір файлу, щоб переконатися, що він не перевищує максимальний дозволений розмір, і лише після цього завантажує файл на локальний комп’ютер.

Перше, що виконується в функції download_file, підключення до SFTP-сервера через протокол paramiko.

`paramiko.Transport()` — ця функція ініціалізує з’єднання з SFTP-сервером за допомогою хоста і порту. У даному випадку sftp_host і sftp_port — параметри, які визначають, до якого сервера буде підключатися, і через який порт.

`transport.connect()` — виконується аутентифікація з використанням ім’я користувача та пароля для входу на сервер.

`paramiko.SFTPClient.from_transport(transport)` — створюється SFTP-клієнт, який використовує активне з’єднання (transport) для роботи з файлами на сервері.

Підключення дає змогу здійснювати операції з файлами на віддаленому сервері, такі як завантаження, переміщення чи видалення файлів (лістинг 2.4).

Лістинг 2.4 — Підключення до SFTP-сервера

```
    print(f"🔗 Підключаюсь до {sftp_host}...")
    transport = paramiko.Transport((sftp_host, sftp_port))
    transport.connect(username=sftp_username, 
password=sftp_password)
```

```
sftp = paramiko.SFTPClient.from_transport(transport)
```

Після підключення необхідно сформувати повні шляхи до файлів на сервері та на локальному диску.

`remote_file_path` — шлях до файлу на SFTP-сервері. Він формується шляхом з'єднання базової директорії на сервері (remote_directory) і імені файлу, який передається в функцію (file_name).

`local_file_path` — шлях до місця на локальному комп'ютері, де буде збережено файл. Шлях формується шляхом з'єднання локальної директорії (local_directory) з іменем файлу (лістинг 2.5).

Лістинг 2.5 — Визначення шляху до файлів

```
remote_file_path = f"{remote_directory}/{file_name}"
local_file_path = os.path.join(local_directory, file_name)
```

Перед тим як завантажити файл, потрібно перевірити його метадані**,** зокрема, його розмір. Дозволяє здійснити перевірку, чи не перевищує файл максимальний ліміт розміру, встановлений для завантаження.

`sftp.stat(remote_file_path)` — отримує метадані для файлу на SFTP-сервері. Ці метадані включають розмір файлу, дату останньої модифікації, власника файлу та інші характеристики.

`file_stat.st_size` — доступ до атрибута st_size метаданих, що повертає розмір файлу в байтах (лістинг 2.6).

Лістинг 2.6 — Отримання метаданих файлу

```
file_stat = sftp.stat(remote_file_path)
file_size = file_stat.st_size
```

Перед завантаженням файлу необхідно перевірити, чи не перевищує його розмір максимальний ліміт, який був визначений раніше.

`if file_size <= MAX_FILE_SIZE_BYTES` — перевіряється, чи розмір файлу не перевищує ліміт. Якщо файл має розмір менший або рівний максимально дозволеному (у байтах), він буде завантажений.

`with open(local_file_path, "wb") as f` — відкривається локальний файл для запису в бінарному режимі ("wb"), де f є об'єктом файлу, в який буде записано завантажені дані.

`sftp.getfo(remote_file_path, f)` — метод getfo виконує безпосереднє завантаження вмісту файлу з віддаленого сервера на локальний диск.

Якщо файл успішно завантажений, виводиться повідомлення "Файл успішно завантажено"(лістинг 2.7).

Лістинг 2.7 — Перевірка розміру файлу

```
if file_size <= MAX_FILE_SIZE_BYTES:
    with open(local_file_path, "wb") as f:
        sftp.getfo(remote_file_path, f)
    print(f"Файл {file_name} успішно завантажено.")
else:
    print(f"Файл {file_name} перевищує максимальний розмір.")
```

Після завантаження файлу необхідно закрити підключення до SFTP-сервера та завершити сесію.

`sftp.close()` — закриває SFTP-з'єднання, що забезпечує безпеку та правильне завершення операцій з файлами на сервері.

`transport.close()` — закриває основне підключення до сервера (лістинг 2.8).

Лістинг 2.8 — Закриття з’єднання

```
sftp.close()
transport.close()
```

Якщо під час виконання функції виникає будь-яка помилка (проблеми з підключенням до сервера, недоступність файлу або інші помилки при завантаженні), вона буде перехоплена через блок try-except, і виведеться відповідне повідомлення.

`except Exception as e` — ловить всі помилки, що можуть виникнути під час виконання коду в блоці try.

`print(f"Помилка при підключенні або завантаженні {file_name}: {e}")` — виводить повідомлення про помилку з описом проблеми (лістинг 2.9).

Лістинг 2.9 — Обробка помилок

```
    except Exception as e:
        print(f"Помилка при підключенні або завантаженні
{file_name}: {e}")
```

Завжди потрібно переконатися, що на сервері є доступні файли для завантаження. Робиться за допомогою sftp.listdir(), що повертає список файлів в зазначеній директорії.

`sftp.listdir()` — отримує список усіх файлів у вказаній директорії на сервері (лістинг 2.10).

Лістинг 2.10 — Перевірка наявності файлів на сервері

```
    def execute(current_date: str):
        transport = paramiko.Transport((sftp_host, sftp_port))
        transport.connect(username=sftp_username,
password=sftp_password)
        sftp = paramiko.SFTPClient.from_transport(transport)
        sftp.chdir(remote_directory)
        files = sftp.listdir()
        if files:
            for file in files:
                download_file(file)
        else:
            print("⚠️ На сервері немає файлів для завантаження.")
        sftp.close()
        transport.close()
```

У Додатку Г представлено Python-скрипт, що виконує підключення до SFTP-сервера, завантаження файлів та їх подальше локальне збереження.

У Додатку В.1 подано структуру цього скрипту, яка ілюструє основні етапи обробки: встановлення з'єднання, автентифікацію, завантаження файлів та їх збереження на локальному диску.

### 2.1.2 Вибір методів їх обробки вхідних файлів

Процес обробки завантажених файлів включає кілька важливих етапів, кожен з яких відповідає за правильну підготовку даних для подальших операцій у ETL-системі.

Першим етапом є отримання шляху до папки, де зберігаються завантажені файли. Дозволяє визначити, де знаходяться всі файли, і далі працювати з ними.

`os.path.expanduser("~")` — ця функція отримує шлях до домашньої директорії поточного користувача. Дає можливість динамічно працювати з домашніми директоріями на різних системах.

`os.path.join(home_dir, "Downloads")` — тут використовуємо os.path.join, щоб коректно об'єднати шляхи до папки Downloads (завантаження), не залежно від операційної системи.

`datetime.today().strftime('%Y-%m-%d')` — ця частина отримує поточну дату у форматі YYYY-MM-DD, щоб сформувати унікальну директорію для зберігання файлів на кожен день.

Цей крок дозволяє створювати чітко організовану структуру файлів, де дані зберігаються за днями, що дозволяє легко знаходити та обробляти файли з конкретної дати (лістинг 2.11).

Лістинг 2.11 — Обробка завантажених файлів

```
def get_today_folder():
    home_dir = os.path.expanduser("~")
    downloads_dir = os.path.join(home_dir, "Downloads")
    today = datetime.today().strftime('%Y-%m-%d')
    folder_path = os.path.join(downloads_dir, today)
    return folder_path
```

Наступним етапом є отримання всіх CSV-файлів в зазначеній директорії, оскільки потрібно працювати лише з ними. Використовуємо функцію os.listdir(), щоб отримати всі файли в директорії, та фільтруємо їх за розширенням .csv.

`os.path.exists(input_dir)` — перевірка на існування директорії. Якщо папка не існує, функція повертає порожній список.

`os.listdir(input_dir)` — отримує список всіх файлів і папок у вказаній директорії.

`f.endswith('.csv')` — фільтрує тільки ті файли, що мають розширення .csv, оскільки лише такі файли містять необхідні дані для подальшої обробки.

`os.path.join(input_dir, f)` — формує повний шлях до кожного файлу (лістинг 2.12).

Лістинг 2.12 — Отримання списку CSV-файлів для обробки

```
def get_today_files(input_dir):
    """Отримує всі CSV-файли за сьогоднішню дату у вхідній папці."""
    if not os.path.exists(input_dir):
        print(f"Папка {input_dir} не існує!")
        return []
    all_files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith('.csv')]
    return all_files
```

Коли отримали список файлів, наступним етапом є фільтрація даних. Важливий етап, оскільки файли можуть містити некоректні дані або непотрібні рядки, що потрібно відсіяти, щоб підготувати дані до подальшого використання.

Фільтрація в даному контексті означає перевірку:

— чи є дані в кожному файлі;

— чи містять файли обов'язкові поля;

— чи відповідають числові значення правильному формату.

`pd.read_csv(file_path)` — за допомогою pandas завантажуємо дані з CSV-файлу в DataFrame, що є зручним форматом для подальшої роботи з даними.

`df.empty` — перевірка, чи не порожній файл. Якщо файл не містить жодного рядка даних, його обробка припиняється (лістинг 2.13).

Лістинг 2.13 – Фільтрація та перезапис файла

```
def filter_csv(file_path):
    """Фільтрує дані у файлі та перезаписує його."""
    if not os.path.exists(file_path):
        print(f" Файл {file_path} не знайдено, пропускаємо.")
        return

    try:
        df = pd.read_csv(file_path)

        if df.empty:
            print(f" Файл {file_path} порожній, пропускаємо.")
            return
```

Фільтрація даних включає наступний крок:

`groupby()` — групує дані за певними полями (за платформою, напрямком або іншими параметрами), у цьому випадку дані групуються за напрямком та платформою;

`head(3)` — для кожної групи вибираються перші 3 записи, якщо їх більше трьох;

`pd.concat(filtered_rows,ignore_index=True)`— об'єднує всі відфільтровані записи в один DataFrame для подальшого збереження (лістинг 2.14).

Лістинг 2.14 – Фільтрація даних

```
    def slice_data_by_fields(group):
        filtered_rows = []
        one_way_group = group[group["is_one_way"] == "Yes"]
        two_way_group = group[group["is_one_way"] == "No"]
        for (origin, destination), subgroup in
one_way_group.groupby(["origin", "destination"]):
            desktop_group = subgroup[subgroup["platform"] ==
"desktop"]
            app_group = subgroup[subgroup["platform"] == "app"]
            if len(desktop_group) >= 3:
                filtered_rows.append(desktop_group.head(3))
            if len(app_group) >= 3:
                filtered_rows.append(app_group.head(3))
        for (origin, destination), subgroup in
two_way_group.groupby(["origin", "destination"]):
            desktop_group = subgroup[subgroup["platform"] ==
"desktop"]
            app_group = subgroup[subgroup["platform"] == "app"]
            if len(desktop_group) >= 3:
                filtered_rows.append(desktop_group.head(3))
            if len(app_group) >= 3:
                filtered_rows.append(app_group.head(3))
        return pd.concat(filtered_rows, ignore_index=True) if
filtered_rows else pd.DataFrame()
```

Після того як дані були відфільтровані та оброблені, наступним кроком є перезапис файлу з новими, відфільтрованими даними. Дозволяє зберегти тільки коректні записи в файлі, що підготовлені для подальшої обробки або аналізу.

`filtered_data.to_csv(file_path, index=False`) — перезаписує файл з новими, відфільтрованими даними. Параметр index=False вказує на те, щоб індекси рядків не зберігались у файлі.

У Додатку Д наведено Python-скрипт, який автоматизує процес обробки CSV-файлів, завантажених за поточну дату. Скрипт виконує пошук файлів у відповідній папці, фільтрацію даних відповідно до заданих умов та перезапис відфільтрованих результатів.

У Додатку В.2 подано структуру цього процесу, що демонструє основні кроки: виявлення файлів, перевірку дати, обробку даних та збереження результатів.

## 2.2 Вибір технологічного стеку для побудови ETL-системи

### 2.2.1 Pandas — бібліотека для обробки, трансформації та аналізу даних у форматі CSV

Pandas — пакет обробки даних у Python для табличних даних. Тобто дані у формі рядків і стовпців, відомі як DataFrames. Інтуїтивно зрозуміло, що можна сприймати DataFrame як аркуш Excel.

Функціональні можливості pandas включають перетворення даних, як-от сортування рядків і взяття підмножин, для обчислення підсумкових статистичних даних, таких як середнє значення, зміни форми DataFrames і об’єднання DataFrames разом. Pandas добре працює з іншими популярними пакетами даних Python, які часто називають екосистемою PyData, включно з:

— NumPy для чисельних обчислень;

— Matplotlib , Seaborn , Plotly та інші пакети візуалізації даних;

— scikit-learn для машинного навчання.

Pandas використовується в усьому робочому процесі аналізу даних. Що можна зробити за допомогою pandas :

— імпортувати набори даних із баз даних, електронних таблиць, файлів зі значеннями, розділеними комами (CSV) тощо;

— очистити набори даних;

— упорядкувати набори даних, змінивши їхню структуру на відповідний формат для аналізу;

— агрегувати дані шляхом обчислення підсумкових статистичних даних, таких як середнє значення стовпців, кореляція між ними тощо;

— візуалізувати набори даних і виявлення ідеї.

Pandas містить функції для аналізу часових рядів і аналізу текстових даних.

Безсумнівно, pandas є потужним інструментом обробки даних, який має кілька переваг, зокрема:

— створено для Python: Python є найпопулярнішою у світі мовою машинного навчання та науки про дані;

— менш багатослівний на одиничні операції: код, написаний у pandas, є менш докладним, тому для отримання бажаного результату потрібно менше рядків коду;

— інтуїтивно зрозумілий перегляд даних: pandas пропонує винятково інтуїтивне представлення даних, що полегшує розуміння та аналіз даних;

— розширений набір функцій: він підтримує великий набір операцій від дослідницького аналізу даних, роботи з відсутніми значеннями, обчислення статистики, візуалізації однофакторних і двофакторних даних і багато іншого;

— працює з великими даними: pandas легко обробляє великі набори даних, він пропонує швидкість і ефективність під час роботи з наборами даних порядку мільйонів записів і сотень стовпців, залежно від машини [14].

### 2.2.2 PostgreSQL — об'єктно-реляційна система керування базами даних для зберігання нормалізованих та агрегованих даних

PostgreSQL — передова система реляційних баз даних корпоративного класу з відкритим кодом. PostgreSQL підтримує як SQL (реляційні), так і JSON (нереляційні) запити.

PostgreSQL — високостабільна база даних, розроблена спільнотою з відкритим вихідним кодом протягом понад 20 років.

PostgreSQL використовується як основна база даних для багатьох веб-додатків, а також мобільних і аналітичних додатків [15].

Спочатку проект називався POSTGRES, про старішу базу даних Ingres, якбула розроблена в Берклі. Метою проекту POSTGRES було додати мінімальні функції, необхідні для підтримки кількох типів даних.

У 1996 році проект POSTGRES було перейменовано на PostgreSQL, щоб чітко проілюструвати його підтримку SQL. Сьогодні PostgreSQL прийнято скорочувати як Postgres.

Відтоді PostgreSQL Global Development Group, віддана спільнота учасників, продовжує випускати безкоштовний проект бази даних з відкритим кодом.

Спочатку PostgreSQL був розроблений для роботи на UNIX-подібних платформах. А потім PostgreSQL розвинувся для роботи на різних платформах, таких як Windows, macOS і Solaris [17].

PostgreSQL підтримує декілька вбудованих механізмів для завантаження даних з текстових файлів у форматі CSV, які є стандартом де-факто для обміну табличними даними. Формат CSV (Comma-Separated Values) — структурований текстовий формат, у якому значення поділяються роздільниками, зазвичай комами або крапками з комою. Завдяки цьому формату забезпечується висока сумісність із електронними таблицями, базами даних та іншими засобами обробки даних (PostgreSQL Docs [18]).

Основним інструментом PostgreSQL для завантаження даних є команда COPY, яка дозволяє імпортувати дані безпосередньо з файлу у таблицю бази даних. Цей метод виконується на стороні сервера, і файл має бути доступним файловій системі сервера. Крім цього, клієнтська утиліта psql підтримує команду \copy, яка виконує подібну функцію, але з боку клієнта, що є зручним для завантаження файлів, розміщених локально (PNS ХНЕУ [16]).

Під час завантаження CSV-файлів у PostgreSQL можуть бути використані додаткові параметри, такі як HEADER (наявність заголовків у файлі), DELIMITER (вибір роздільника), ENCODING (встановлення кодування) та NULL (визначення значень, які трактуються як NULL). Дозволяє адаптувати процес імпорту до формату початкових даних та забезпечити коректність записів (PostgreSQL Docs ) [17].

Важливим аспектом є підготовка таблиці, у яку імпортуються дані. Вона повинна відповідати структурі CSV-файлу, зокрема кількістю та порядком стовпців, типами даних і наявністю ключових обмежень. Для забезпечення

цілісності інформації після імпорту можна використовувати додаткові перевірки, тригери або збережені процедури (DKPKM) [18].

У випадку великого обсягу даних доцільним є використання матеріалізованих представлень або тимчасових таблиць, у які спочатку імпортуються дані для подальшої перевірки та трансформації перед остаточним завантаженням у основну базу (PostgreSQL Docs) [15].

Таким чином, PostgreSQL забезпечує гнучкий, потужний і безпечний механізм для завантаження CSV-файлів, що робить її ефективним інструментом у завданнях інтеграції, аналізу та вивантаження даних.

2.2.3 Apache Airflow — платформа для створення, оркестрації та моніторингу робочих процесів (workflows).

Airflow — платформа оркестровки пакетного робочого процесу з відкритим кодом. Він призначений для програмного створення, планування та моніторингу робочих процесів. Розширювана структура Python Airflow дозволяє користувачам створювати робочі процеси, підключаючись практично до будь-якої технології.

Спочатку розроблений на Airbnb Максимом Бошеменом у жовтні 2014 року, Airflow отримав широке поширення в спільнотах інженерії даних і науки про дані. Airflow був із відкритим вихідним кодом із самого першого коміту та офіційно включений у Airbnb GitHub, про що було оголошено у червні 2015 року.

2.2.3.1 Внутрішня робота Airflow

Apache Airflow — інструмент оркестровки робочого процесу з відкритим кодом. Його розширюваний і масштабований характер робить його кращим вибором серед спеціалістів із обробки даних. Він допомагає виконувати задачі розробки даних, ефективно керувати процесами ETL і керувати конвеєрами даних. В основі Airflow лежать спрямовані ациклічні графіки (DAG). DAG — це, по суті, робочі процеси, які визначають послідовність завдань і їхні залежності. Користувачі можуть використовувати код Python для визначення цих DAG, що робить роботу з нею легкою. Робочий процес може бути настільки простим чи

складним, як того вимагає ваш сценарій використання, від обробки даних до керування моделями машинного навчання. Airflow розроблено для масштабування та розвитку відповідно до потреб. Він пропонує зручний інтерфейс і надійні API для керування складними даними та залежностями. Решта API можна використовувати для взаємодії з системою та автоматизації завдань. Архітектура Airflow розроблена для керування робочими процесами в режимі реального часу, що робить її неймовірно ефективною. AWS (вебслужба Amazon), Azure, Google Cloud та інші хмарні платформи підтримують Airflow, доводячи її універсальність. Його можна легко інтегрувати з іншими інструментами Apache Software Foundation, такими як Spark і Hadoop, та іншими системами, такими як SQL, Hive і GitHub. Розширюваність Apache Airflow також поширюється на його можливості оповіщення. Airflow плавно інтегрується зі Slack для оповіщення в реальному часі. Крім того, такі постачальники, як Astronomer, спрощують керовані робочі процеси.

2.2.3.2 Розуміння архітектури повітряного потоку

Робочі процеси Airflow представлені як DAG, де кожен вузол на графіку представляє завдання, а краї визначають залежності між завданнями. Самі ці завдання описують, що робити, будь то отримання даних, запуск аналізу, запуск інших систем тощо.

Основні компоненти Airflow:

— планувальник: запускає заплановані робочі процеси та надсилає завдання виконавцю;

— виконавець: виконує завдання, переважно за допомогою робітників;

— вебсервер: надає інтерфейс користувача (UI) для перевірки, запуску та налагодження поведінки та завдань DAG;

— папка з файлами DAG: зчитується планувальником і виконавцем (і будь-якими працівниками, які має виконавець);

— база даних метаданих: використовується планувальником, виконавцем і веб-сервером для збереження стану (рис. 2.1).

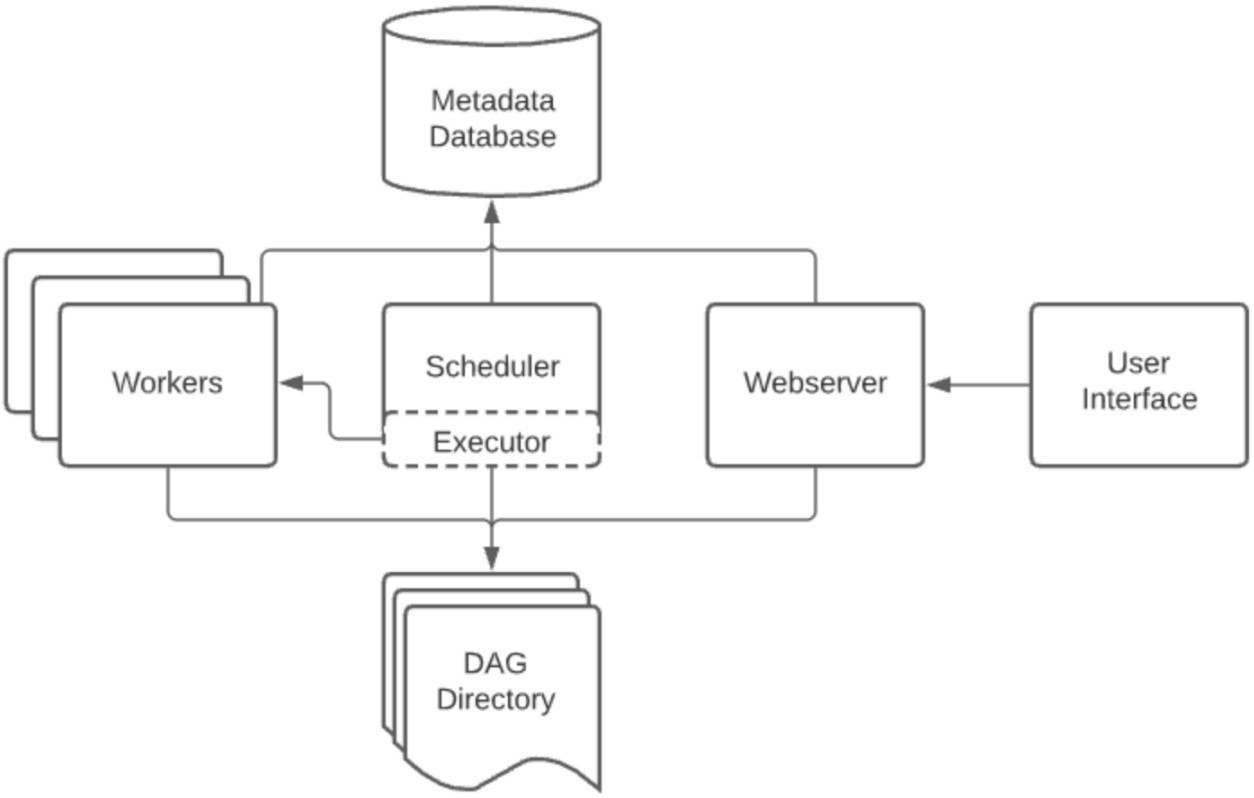



Рис. 2.1 — Схема повітряного потоку Apache

Основні поняття Airflow:

— DAG: збирає завдання, встановлюючи їхні залежності та зв’язки, щоб диктувати послідовність їх виконання в робочому процесі Airflow;

— запуски DAG: кожного разу, коли запускається DAG, генерується запуск DAG, і запускаються всі завдання, укладені в ньому, стан цього запуску DAG залежить від станів завдань, кожен запуск DAG працює незалежно від інших, тому кілька запусків DAG можуть виконуватися одночасно;

— завдання: є основною одиницею виконання в Airflow, завдання організовуються в DAG, а потім встановлюються залежностями вгорі та внизу, щоб диктувати послідовність, у якій вони повинні виконуватися;

— оператори: шаблон для попередньо визначеного завдання, яке можна декларативно визначити в DAG.

Деякі з поширених основних операторів є:

— BashOperator — виконує команду bash;

— PythonOperator — викликає довільну функцію Python;

— EmailOperator — надсилає електронний лист;

— декоратор @task — виконує довільну функцію Python (рекомендується замість класичного PythonOperator для виконання викликів Python без відтворення шаблону в її аргументах), окрім основних операторів, можна встановити багато інших операторів із пакетів постачальників спільноти, e,g, MySqlOperator, PostgresOperator, HiveOperator, SnowflakeOperator тощо;

— датчики: унікальна різноманітність операторів у Airflow, які сконструйовані таким чином, щоб чекати, доки не будуть виконані певні умови, коли ці умови досягнуті, вони позначаються як успішні, ініціюючи виконання наступних завдань у групах DAG, датчики можуть працювати в двох різних режимах, у режимі за замовчуванням 'poke' робочий слот зайнятий протягом повного часу роботи датчика, другий режим, «перепланування», є більш ефективним, оскільки він займає робочий слот лише під час перевірки процесу, а потім перебуває в режимі сну протягом встановленої тривалості між перевірками;

— TaskFlow (потік завдань): корисний для створення чітких і зрозумілих DAG, особливо коли більшість DAG складаються з базового коду Python, а не з операторів, робиться за допомогою декоратора @task. TaskFlow піклується про переміщення входів і виходів між вашими завданнями за допомогою XCom, а також автоматично обчислює залежності;

— планувальники: контролює всі завдання та DAG, ініціюючи завдання, щойно їх залежності будуть виконані з певним інтервалом (за замовчуванням одна хвилина) планувальник переглядає результати груп DAG, визначаючи, чи будь-яке з активних завдань готове до виконання в конвеєрах даних;

— виконавці: для виконання завдань потрібні виконавці, ці виконавці мають загальний API, що дозволяє легко замінювати на основі конкретних вимог вашої установки, виконавці можна розділити на два типи: ті, які виконують завдання локально, і ті, які призначені для виконання завдань віддалено;

— XComs: скорочення від крос-комунікацій, як випливає з назви, вони дозволяють спілкуватися між різними завданнями (за замовчуванням завдання повністю ізольовані та можуть виконуватися на іншій машині);

— змінні: глобальне сховище ключів і значень, до якого можна запитувати із завдань Airflow на відміну від XCom, які використовуються для передачі даних між завданнями, змінні слід використовувати лише для глобальних конфігурацій;

— параметри: служать методом доставки конфігурацій часу виконання для завдань у робочих процесах, параметри за замовчуванням можна налаштувати в коді DAG, можна надати додаткові параметри або змінити значення за замовчуванням під час виконання під час запуску DAG.

Apache Airflow пропонує кілька ключових переваг у сфері обробки даних і керування робочим процесом.

Airflow забезпечує платформу для оркестрування складних робочих процесів, дозволяючи користувачам визначати, планувати та контролювати завдання скоординовано та ефективно. Airflow DAG дозволяють створювати динамічні робочі процеси, пропонуючи адаптацію до мінливих вимог.

Завдяки DSL (доменно-орієнтована мова) на основі Python Airflow пропонує гнучкість у вираженні робочих процесів. Його модульна архітектура дозволяє користувачам розширювати функціональні можливості шляхом створення спеціальних операторів і датчиків.

Оскільки вимоги до обробки даних зростають, Airflow масштабується горизонтально, дозволяючи організаціям обробляти зростаючі навантаження та виконувати завдання паралельно. Ця масштабованість гарантує, що система може адаптуватися до мінливих потреб робочих процесів даних.

Airflow заохочує створення модульних багаторазових завдань і робочих процесів. Сприяє більш зручній у супроводі та організованій кодовій базі, зменшуючи надмірність і полегшуючи керування та оновлення робочих процесів з часом.

Airflow легко інтегрується з різними зовнішніми системами, базами даних і хмарними службами. Робить його універсальним вибором для організацій з різноманітними стеками технологій.

Будучи проектом з відкритим кодом, Apache Airflow отримує переваги від активної та зростаючої спільноти. Ця підтримка спільноти призводить до регулярних оновлень, внесків і розробки плагінів, що робить Airflow універсальним інструментом із багатою екосистемою.

Apache Airflow — універсальний інструмент із широким спектром варіантів використання в галузі інженерії даних, науки про дані тощо.

Airflow широко використовується для оркестрування робочих процесів ETL, автоматизації вилучення, перетворення та завантаження даних із різних джерел, таких як ваша CRM, облікові записи в соціальних мережах, ваші облікові записи реклами тощо, у сховище даних або інші місця призначення.

Airflow використовується для автоматизації та планування завдань бізнес-аналітики, таких як вилучення даних, створення звітів і оновлення інформаційної панелі. Окрім робочих процесів даних, Airflow може автоматизувати завдання, пов’язані з інфраструктурою, керування хмарними ресурсами, надання та виведення з експлуатації серверів.

Автоматизовані робочі процеси створення та доставки звітів можна організовувати за допомогою Airflow, що дозволяє організаціям планувати та розповсюджувати звіти на основі попередньо визначених часових рамок.

Airflow можна налаштувати для моніторингу різноманітних процесів, пов’язаних з даними, і ініціювання попереджень або сповіщень у разі проблем або збоїв, підвищуючи надійність системи.

Airflow можна використовувати для забезпечення дотримання політики управління даними та відповідності вимогам шляхом автоматизації робочих процесів, які перевіряють якість даних, походження та безпеку [19].

Apache Airflow — ключове рішення в області обробки даних, яке надає організаціям потужну та гнучку платформу для організації, автоматизації та оптимізації складних робочих процесів. Його динамічні спрямовані ациклічні графіки (DAG) у поєднанні з розширеними функціями планування, моніторингу та розширюваності дають користувачам можливість ефективно керувати різноманітними завданнями — від процесів ETL і міграції даних до конвеєрів

машинного навчання тощо. Підсумовуючи, незалежно від того, початківець ви чи досвідчений розробник даних, надійні функції та масштабована архітектура Airflow обов'язково додадуть переваги вашим потребам у обробці даних [20].

## 2.3 Розробка архітектури системи обробки авіаційних даних

У межах реалізації інформаційної системи було розроблено архітектуру ETL-системи, яка базується на принципах модульності, масштабованості та логічного поділу відповідальності між компонентами. Побудована архітектура забезпечує стабільне та автоматизоване вилучення, обробку й збереження авіаційних даних, що надходять у форматі CSV з агрегаторів авіаперевезень.

Загальна логіка функціонування охоплює чотири основні функціональні складові. Джерело даних — віддалений SFTP-сервер, на якому щодня зберігаються файли з новими записами. Дані надходять у форматі CSV та містять відомості про рейси, аеропорти, перевізників, тарифи, ціни, дати вильотів і прибуттів тощо. Початковий етап, з якого стартує увесь цикл обробки.

Система оркестрації — Apache Airflow, яка виконує автоматичне планування та запуск ETL-процесів. Створений DAG-граф відповідає за послідовне виконання завдань: перевірку наявності нових файлів, зчитування, трансформацію та збереження результатів. Завдяки Airflow забезпечено контроль за залежностями між задачами, обробку помилок, повторні спроби та логування.

Середовище обробки — мова програмування Python із використанням бібліотеки Pandas, що виконує обробку вхідних CSV-файлів.

На етапі обробки реалізовано основні елементи трансформації даних:

— фільтрація записів відповідно до заданих критеріїв;

— стандартизація назв полів, типів даних та форматів дат;

— логічна перевірка відповідності даних специфікації;

— підготовка структурованих таблиць до імпорту.

— система збереження даних — реляційна база даних PostgreSQL, у яку завантажуються трансформовані дані, вона дозволяє зберігати інформацію у відповідності до заздалегідь визначеної схеми таблиць для подальшої аналітики.

На рисунку 2.2 схематично зображено взаємодію між компонентами: щоденне ініціювання Airflow завантажує файли з SFTP-сервера, обробляє їх за допомогою Pandas і передає результати у PostgreSQL. Така побудова архітектури дозволяє:

— автоматизувати щоденне оновлення даних;

— забезпечити чітке логування та контроль якості на кожному етапі;

— легко масштабувати систему шляхом розширення DAG або структури бази;

— адаптувати рішення до нових джерел або форматів даних без значних змін у загальній логіці системи.

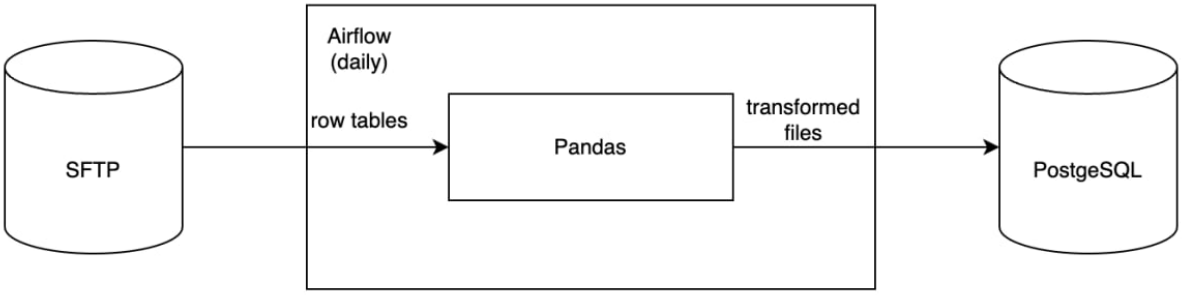



Рисунок 2.2 — Загальна архітектура ETL-системи

Таким чином, запропонована архітектура забезпечує ефективну побудову ETL-системи у контексті обробки авіаційних даних і слугує надійною основою для подальшого розвитку інформаційної системи.

## 2.4 Реалізація DAG-графа в Apache Airflow

Apache Airflow — потужний інструмент для оркестрації робочих процесів. Він використовує DAG для організації та керування задачами, що виконуються в певній послідовності. В даному випадку створено DAG, який автоматизує процес

ETL для обробки авіаційних даних. Цей процес включає завантаження файлів з SFTP-сервера, їх обробку, форматування та завантаження в PostgreSQL.

Для створення DAG в Airflow необхідно імпортувати стандартні модулі для роботи з часом, а також PythonOperator для виконання Python-функцій в рамках DAG.

`sys.path.append()` дозволяє додавати додаткові шляхи до папок, де знаходяться модулі.

Імпортовані функції виконують різні операції на етапах ETL: завантаження файлів, обробка, форматування та завантаження в базу даних.

Налаштування параметрів DAG включає в себе визначення параметрів виконання завдань, таких як кількість повторних спроб, час початку виконання та інтервал між спробами.

`owner` — вказує на того, хто володіє DAG.

`depends_on_past` — якщо встановлено в False, кожен запуск DAG не залежатиме від результатів попереднього.

`start_date` — визначає, коли DAG почне свою роботу (зазначено з 1 січня 2024).

`retries` — кількість спроб повторного запуску в разі невдачі завдання.

`retry_delay` — інтервал часу між спробами повторного виконання завдання (лістинг 2.15).

Лістинг 2.15 — Налаштування параметрів DAG

```
default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2024, 1, 1),
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}
```

У Airflow можна передавати дані між різними завданнями за допомогою XCom. Функція `process_files_from_xcom` отримує список файлів з XCom і викликає функцію `filter_csv` для їх обробки.

`task_instance.xcom_pull()` — отримує список файлів, який передається від попереднього завдання download_files через XCom.

Функція filter_csv викликається для обробки кожного завантаженого файлу (лістинг 2.16).

Лістинг 2.16 — Реалізація обробки кожного файлу

```
def process_files_from_xcom(task_instance):
    """Отримує список файлів із XCom і обробляє кожен із них."""
    file_paths = task_instance.xcom_pull
(task_ids="download_files", key="downloaded_files")

    if not file_paths:
        print("⚠️ Немає файлів у XCom для обробки.")
        return

    for file_path in file_paths:
        filter_csv(file_path)
```

Після налаштування параметрів, наступним етапом є створення самого DAG, який буде визначати послідовність завдань, їх час виконання та залежності.

`dag_id` — унікальний ідентифікатор для DAG.

`default_args` — передаємо загальні параметри для всіх завдань.

`schedule="@daily"` — визначає, що DAG буде виконуватися щодня.

`catchup=False` — гарантує, що DAG не виконуватиме пропущені виконання завдань (лістинг 2.17).

Лістинг 2.17 — Створення DAG

```
with DAG(
    dag_id="sftp_to_postgres",
    default_args=default_args,
    schedule="@daily",  # Запуск щодня о 00:00
    catchup=False,
) as dag:
```

Кожне завдання в Airflow — окремий етап у ETL-системі. Створюються чотири основні завдання:

Завантажує файли з SFTP-сервера через функцію `execute`, передаючи поточну дату `({{ ds }})` для завантаження відповідних файлів.

Викликає функцію `process_files_from_xcom` для обробки файлів, отриманих через `XCom`.

Завдання для форматування оброблених даних перед їх завантаженням. Завдання для обробки та завантаження файлів у PostgreSQL (лістинг 2.18.).

Лістинг 2.18 — Реалізація завдань DAG

```
# 🛠️ 1**Завантаження файлів з SFTP**
    download_task = PythonOperator(
        task_id="download_files",
        python_callable=execute,
        op_args=["{{ ds }}"]
    )
    # 2 **Сортування файлів**
    sort_task = PythonOperator(
        task_id="sort_file",
        python_callable=process_files_from_xcom,
        provide_context=True
    )
    # 3 **Форматування виводу**
    output_task = PythonOperator(
        task_id="output_file",
        python_callable=format_output_function
    )
    # 4**Обробка файлів і завантаження у PostgreSQL**
    process_task = PythonOperator(
        task_id="process_and_load_files",
        python_callable=process_and_load_data
    )
```

Визначення залежностей між завданнями є важливою частиною DAG**.** Завдання виконуються по черзі. Спочатку виконується `download_task` для завантаження файлів, потім — `sort_task` для їх обробки, `output_task` для форматування даних і, нарешті, `process_task` для завантаження в базу даних.

```
download_task >> sort_task >> output_task >> process_task
```

Додаток Е містить лістинг DAG-графа для побудови ETL-системи, що включає отримання даних через SFTP-протокол, їхню обробку та завантаження у реляційну базу даних.

## 2.5 Проєктування структури таблиці у PostgreSQL

Проєктування структури таблиці у PostgreSQL є важливим етапом побудови ETL-системи, оскільки саме ця таблиця виступає як основне сховище для

оброблених авіаційних даних, отриманих з агрегаторів. Основною вимогою до моделі було забезпечення гнучкого збереження як коректних, так і некоректних записів без втрати жодного значення.

Зважаючи на специфіку вхідних CSV-файлів, що можуть містити пропущені значення, відхилення від формату або змішані типи, було прийнято рішення використовувати текстовий тип даних (text) для всіх полів таблиці. Такий підхід дозволяє:

— зберігати будь-які значення незалежно від їх формату;

— уникати збоїв при імпорті невалідних даних;

— забезпечити повноту даних для подальшого аналізу Data Quality-командою;

— гнучко застосовувати валідаційні правила вже на етапі обробки в Python.

Таблиця створена у базі даних ag_air_flights (повітряні польоти) та містить понад 50 колонок, які охоплюють:

— часові мітки: дати вильоту, прильоту, спостереження;

— географічні дані: origin, destination;

— авіаційні характеристики: outbound_stops, outbound_flight_no (номер вихідного борту), fare_basis, carrier (назва перевізника);

— ціни та валюта: price_outbound, price_inc, currency, tax;

— метадані: platform (платформа), sourceOTA (ресурс), rout_type (тип маршруту), screenshot (скріншот), тощо.

Перед тим як реалізовувати процес завантаження оброблених CSV-файлів, необхідно створити середовище зберігання — базу даних, що виступає цільовим репозиторієм у межах архітектури ETL. У даній роботі для цієї мети було обрано PostgreSQL — потужну об’єктно-реляційну систему керування базами даних із відкритим вихідним кодом, що відзначається високою продуктивністю, масштабованістю та підтримкою складних типів даних.

База даних створюється локально на машині користувача за допомогою командного рядка або графічного інтерфейсу ( через pgAdmin). У цьому проєкті використано CLI-інтерфейс: `createdb ag_air_flights`

При підключенні до бази у скрипті використовується обліковий запис користувача victoriashalahan. Для цього попередньо створюється користувач і призначаються права доступу:`createuser victoriashalahan --pwprompt`

`psql -d ag_air_flights -c "GRANT ALL PRIVILEGES ON DATABASE ag_air_flights TO victoriashalahan;"`

Усі поля, включаючи технічні зберігаються у вигляді тексту. Надалі ці дані можуть бути проаналізовані, візуалізовані або використані для побудови додаткових структур (лістинг 2.19).

Лістинг 2.19 — Структура таблиці

```
CREATE TABLE flights_data (
    id text,
    url text,
    check_in_date text,
    check_out_date text,
    observation_date text,
    observation_time text,
    -- інші поля пропущені для прикладу
    comment text
);
```

Завершальним етапом ETL-системи є передача оброблених даних до системи зберігання. У межах реалізованої архітектури роль такого сховища виконує PostgreSQL — об'єктно-реляційна СУБД, що забезпечує зберігання структурованих записів з можливістю подальшої аналітики. Для організації процесу завантаження реалізовано окрему функцію `process_and_load_data()`, яка виконує автоматизоване зчитування CSV-файлів із локальної директорії, їх обробку та імпорт у таблицю `flights_data`.

На початку функції виконуються базові налаштування робочого середовища: імпорт необхідних бібліотек, формування шляху до цільової папки, перевірка її наявності та створення підключення до бази даних.

Етап побудови абсолютного шляху до каталогу, в якому розміщено CSV-файли з авіаційними даними за поточну дату. Підкаталог створюється щоденно у форматі YYYY-MM-DD у межах директорії Downloads, що дозволяє:

— організовано зберігати завантаження по днях;

— автоматизовано ідентифікувати потрібні дані для обробки;

— уникати втрат при повторних завантаженнях (лістинг 2. 20).

Лістинг 2.20 — Формування шляху до робочої директорії

```
downloads_folder = os.path.expanduser("~/Downloads")
today_folder = datetime.today().strftime('%Y-%m-%d')
target_folder = os.path.join(downloads_folder, today_folder)
```

Перед переходом до обробки виконується перевірка існування цільової директорії. У разі її відсутності система не викликає помилку, а коректно завершує виконання функції, зафіксувавши відповідне повідомлення в логах. Такий підхід дозволяє уникнути аварійних зупинок та є важливою умовою побудови стійкого до збоїв автоматизованого пайплайну (лістинг 2. 21).

Лістинг 2.21 — Перевірка наявності каталогу з файлами

```
if not os.path.exists(target_folder):
    log.error(f"❌ Папка {target_folder} не знайдена.")
    return
```

У лог-систему виводиться службове повідомлення про директорію, в якій здійснюється пошук даних. Полегшує аудит роботи скрипту, особливо у разі використання кількох джерел або одночасного оброблення даних за різні дати.

Задання базових параметрів для створення з'єднання з PostgreSQL:

— `db_name` — назва бази даних, у яку здійснюється імпорт записів;

— `username / password` — облікові дані користувача;

— `host` — адреса сервера (у цьому випадку локальний хост);

— `port` — стандартний порт для PostgreSQL (лістинг 2.22).

Лістинг 2.22 — Параметри підключення до PostgreSQL

```
db_name = "ag_air_flights"
username = " username"
password = " password"
host = "localhost"
```

```
port = "5432"
```

Функція `create_engine()` створює об’єкт підключення до PostgreSQL із використанням драйвера `psycopg2`. Через цей об’єкт engine бібліотека Pandas зможе виконувати прямий імпорт DataFrame у таблицю бази даних, що забезпечує ефективну інтеграцію Python-обробки з PostgreSQL (лістинг 2.23).

Лістинг 2.23 — Створення з’єднання через SQLAlchemy

```
engine =
create_engine(f'postgresql+psycopg2://{username}:{password}@{host}:{
port}/{db_name}')
```

У Додатку Ж наведено Python-скрипт, який здійснює пошук CSV-файлів у папці з поточною датою, читає їх за допомогою бібліотеки Pandas та завантажує дані до бази даних PostgreSQL, а у додатку В.3 зображена структура файла.

# 3 ПЕРЕВІРКА ТА ВАЛІДАЦІЯ ETL-СИСТЕМИ ОБРОБКИ АВІАЦІЙНИХ ДАНИХ

## 3.1 Валідація моделі та структури даних

Валідація вхідних даних є невід'ємним етапом побудованого ETL-процесу, оскільки забезпечує надійність, структурну узгодженість і логічну правильність інформації, яка потрапляє у базу даних. Особливо важливо у випадку, коли джерелом даних є сторонні сервіси, а формат — CSV-файл із великою кількістю умовно-обов’язкових полів, як у даному проєкті.

У розробленій специфікації даних описано понад 50 полів, кожне з яких має чітко визначений тип, обов’язковість та залежності від значень інших полів. Така структура вимагає багаторівневої валідації — не лише за типом, а й за логікою залежностей.

Основу валідації становить перевірка наявності обов’язкових полів. Поля, які є строго обов’язковими повинні бути присутні у кожному записі незалежно від умов, їх відсутність є критичною помилкою. Своєчасне виявлення таких помилок дозволяє запобігти потраплянню некоректної інформації до сховища даних, що забезпечує цілісність і стабільну роботу всієї ETL-системи.

Частина полів є умовно обов’язковими — вони мають бути присутні тільки за певних умов. Наприклад:

— check_out_date, inbound_departure_date та інші inbound — поля є обов’язковими лише у випадку, якщо is_one_way = NO (тобто квиток із поверненням);

— outbound_travel_stop_over і outbound_travel_stop_over_duration мають бути заповнені, лише якщо outbound_stops > 0;

— OTA означає «по повітрю» і стосується розповсюдження інформації бездротовим способом, що є важливим та обов’язковим;

— screenshot обов’язкові для platform = desktop (декстоп), app (додаток), а url обов’язковий лише для platform = desktop, адже для перевірки мобільних додатків використовуються емулятори;

Усі ці залежності реалізовано через внутрішні правила валідації у Pandas: умовні перевірки виконуються на основі значень відповідних полів, а в разі невідповідності помилки фіксуються у відповідну колонку в таблиці.

### 3.1.1 Перевірка типів даних та логічна валідація

Кожне поле перевіряється на відповідність зазначеному типу:

— поля дат: check_in_date, observation_date (дата спостереження), outbound_departure_date — у форматі YYYY-MM-DD;

— поля часу: observation_time (час спостреження), outbound_arrival_time — у форматі HH:MM або HH:MM:SS;

— числові значення (price_outbound, SRP_Rank, outbound_stops) — мають бути валідними цілими або десятковими числами;

— текстові значення — обмежені допустимими шаблонами (origin і destination у форматі IATA-кодів, тобто 3 великі літери).

Окрім формальної перевірки, застосовується логічна валідація між пов’язаними полями. Наприклад:

— якщо вказано outbound_stops > 0, то мають бути вказані відповідні поля про зупинку і її тривалість;

— якщо is_one_way = YES, то всі inbound — поля мають бути порожніми;

— якщо дата вильоту пізніше за дату повернення — такий запис вважається нелогічним і фіксується як помилка.

Перед початком роботи з файлом виконується перевірка, чи існує файл за шляхом input_path. Якщо файл відсутній, скрипт виводить повідомлення про помилку і перериває виконання функції, щоб уникнути подальших помилок при обробці неіснуючого файлу (лістинг 3.1).

Лістинг 3.1 — Валідація наявності файлу

```
if not os.path.exists(input_path):
    print(f"Error: File '{input_path}' not found.")
    return
```

Для кожного оброблюваного рядка CSV створюються два додаткові поля:

— zero_one (0/1) — спочатку встановлюється в "1", що означає успішну валідацію, якщо під час перевірки буде виявлено помилки, значення зміниться на "0";

— comment (коментар) — порожній рядок, у який у процесі валідації будуть додаватися текстові повідомлення про знайдені помилки в даних.

Щоб уникнути помилок через різний регістр літер у заголовках стовпців, створюється словник normalized_headers, де ключами є назви полів у нижньому регістрі, а значеннями — їхні реальні імена у файлі. Дозволяє правильно звертатися до полів незалежно від того, чи були вони записані великими чи малими літерами.

Для кожного поля із множини обов'язкових (required_fields) визначається відповідне ім'я поля у CSV за допомогою словника normalized_headers.

Після цього отримується значення поля для поточного рядка. Якщо значення є порожнім ("" після strip()), скрипт визначає, чи є це допустимим винятком (для квитків в один бік або для платформи "app"). Якщо ні, то буде зафіксована помилка (лістинг 3.2).

Лістинг 3.2 — Валідація обов'язкових полів

```
for field in required_fields:
    normalized_field = normalized_headers.get(field.lower(), field)
    value = row.get(normalized_field, "").strip()
```

Для певних полів, що мають містити числові значення (price_outbound, price_inbound, тощо), додатково перевіряється, чи є значення дійсним десятковим числом. Якщо значення не відповідає числовому формату, рядок позначається як некоректний (zero_one = "0"), а в поле comment записується повідомлення про неправильне вилучення даних (лістинг 3.3).

Лістинг 3.3 — Перевірка числових значень

```
    if field in {"price_outbound", "price_inbound",
"price_outbound_USD", "price_inbound_USD"}:
        if not validate_decimal(value):
            row["zero_one"] = "0"
            row["comment"] += f"{field} was extracted incorrectly; "
```

У Додатку К представлено повний програмний код для обробки файлів за поточною датою, їх валідації та оновлення даних у форматі CSV.

У Додатку В.4 зображена структура, що демонструє основні етапи: пошук файлів, перевірку структури, обробку вмісту та збереження оновлених результатів.

Багато полів, згідно зі специфікацією, є опціональними placeholders (Coupon_code, inbound_seats, Bank_Instrument тощо). Валідація гарантує, що їхня відсутність не порушує логіку завантаження, але при цьому контролюється, щоб у них не з'являлися невалідні значення, які можуть порушити схему бази даних.

### 3.1.2 Формування звітності про помилки

На відміну від класичних ETL-систем, де помилкові записи відкидаються або зберігаються в окремі таблиці, у даному проєкті реалізовано стратегію повного збереження вхідних даних у базу PostgreSQL незалежно від результату валідації. Означає, що як коректні, так і потенційно некоректні або неповні записи потрапляють до єдиної цільової таблиці.

Таке рішення обумовлене практикою подальшого аналізу помилок інженерами з якості даних (Data Quality Engineers). Наявність усіх даних в одному місці дозволяє гнучко виконувати ручну перевірку, аналітику, пошук аномалій та виявлення систематичних проблем у вхідному потоці.

Незалежно від результату валідації, запис залишається частиною загальної таблиці. Забезпечує цілісність потоку даних, зберігає повну історію взаємодії системи з вхідним джерелом, що є надзвичайно цінним з точки зору аудиту та контролю.

Під час розробки було враховано особливості вхідних даних, що надходять з SFTP-сервера та мають складну структуру з контекстно обов'язковими полями. Для обробки даних застосовано бібліотеку Pandas, яка виконує базову трансформацію — фільтрацію, сортування, приведення типів та підготовку до завантаження.

Особливу увагу приділено питанню валідації: реалізовано перевірки на відповідність структурі, типам даних, логічним зв'язкам між полями. Незважаючи на наявність помилок у частини записів, усі дані зберігаються в єдиній цільовій

таблиці PostgreSQL без розділення. Обумовлено практичними вимогами аналітики якості даних, що виконується вручну інженерами Data Quality.

Таким чином, створена система забезпечує автоматичне завантаження та збереження інформації незалежно від її ідеальної відповідності формальній моделі. Водночас реалізовані механізми валідації дозволяють ефективно контролювати якість даних і формувати основу для подальшої оптимізації процесу.

## 3.2 Перевірка працездатності та тестування ETL-системи

Перед проведенням тестування працездатності ETL-системи необхідно запустити середовище виконання Apache Airflow, яке відповідає за оркестрацію всіх етапів обробки даних. У межах реалізації використано стандартний механізм запуску Airflow в локальному середовищі за допомогою команди airflow standalone (рис.3.1).

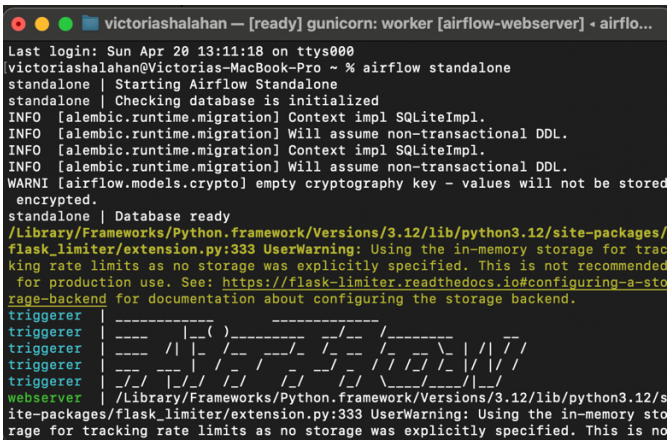



Рисунок 3.1 – Виконання команди airflow standalone

Цей спосіб запуску є зручним для індивідуальної розробки, оскільки поєднує всі базові етапи ініціалізації в одну команду. Зокрема, airflow standalone автоматично:

— створює та ініціалізує базу метаданих (airflow.db);
— запускає планувальник задач (scheduler);
— запускає вебінтерфейс (webserver) на порту 8080;
— створює адміністративного користувача з доступом до інтерфейсу;

— виводить у консоль логін і пароль для доступу до системи.

Після запуску середовища командою airflow standalone необхідно перейти до вебінтерфейсу Apache Airflow для візуального контролю виконання задач.

У вебінтерфейсі користувач отримує доступ до списку доступних DAG-графів. Необхідно переконатися, що створений DAG має статус активний (On) (рис.3.2), після чого його можна ініціювати вручну, натиснувши кнопку «Trigger DAG» (рис.3.3).

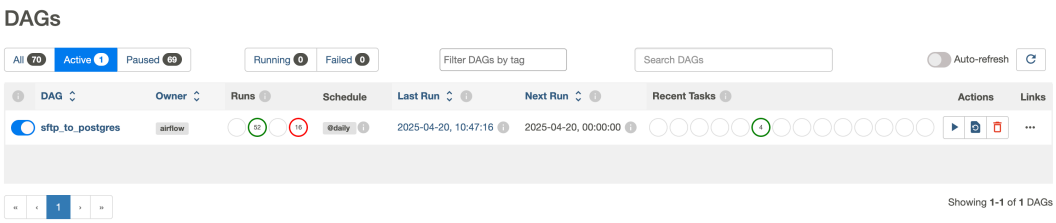



Рисунок 3.2 — Перевірка доступності DAG

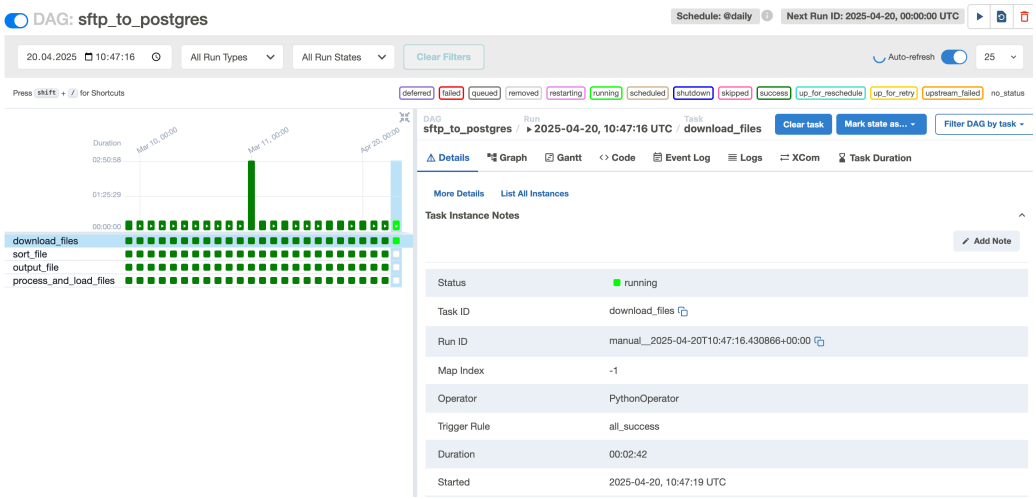



Рисунок 3.3 — Ініціалізація DAG

Після запуску DAG у реальному часі відображається статус виконання кожного окремого таска (download_files, sort_file, process_and_load_files) у вигляді графу залежностей (Graph View) або хронологічної таблиці запусків (Tree View) (рис 3.4).

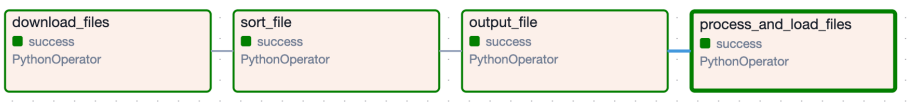



Рисунок 3.4 — Граф залежностей

У разі успішного виконання кожен таск буде позначено зеленим кольором (succes), що свідчить про коректну реалізацію логіки та відсутність помилок у поточному запуску. У випадку помилки — відповідна задача набуде червоного статусу (failed), а користувач зможе переглянути деталізований лог виконання для подальшого аналізу.

Таким чином, вебінтерфейс Airflow відіграє ключову роль у тестуванні: він дозволяє не лише запускати процеси, а й контролювати їх перебіг, аналізувати статуси, повторно запускати окремі етапи, вчасно реагувати на помилки.

Після запуску DAG-графа наступним кроком є перевірка коректності роботи етапів ETL-системи. На цьому етапі необхідно впевнитися, що система правильно згенерувала шлях до директорії з даними відповідно до поточної дати, а також що у вказаній папці дійсно наявні оброблені CSV-файли.

Шлях до папки формується динамічно в межах функції process_and_load_data() на основі поточної дати у форматі YYYY-MM-DD.

Під час виконання скрипта перевіряється наявність цієї директорії. У разі її відсутності скрипт завершить роботу, зафіксувавши повідомлення про помилку у лог-файлі. Таким чином, перевірка існування цільової папки є базовим кроком, що підтверджує коректне формування структури зберігання даних.

Крім того, варто переконатися у наявності хоча б п’яти .csv-файлів всередині відповідної папки (рисунок 3.5). У випадку порожньої директорії функція знову ж таки завершить виконання, повідомивши про це в логах, що дозволяє уникнути фіктивного запуску подальших етапів.

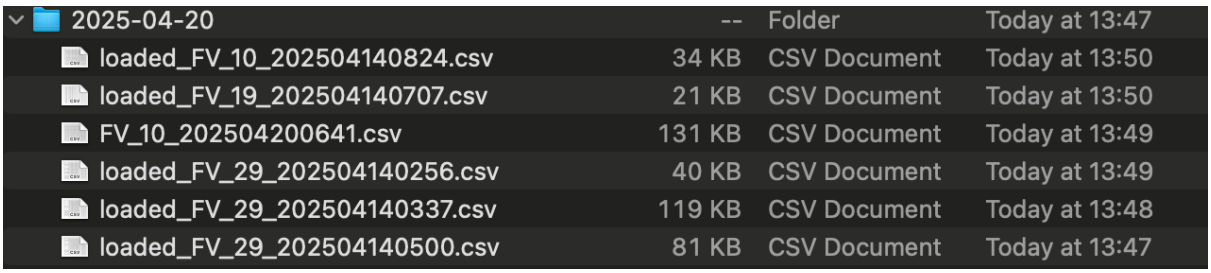
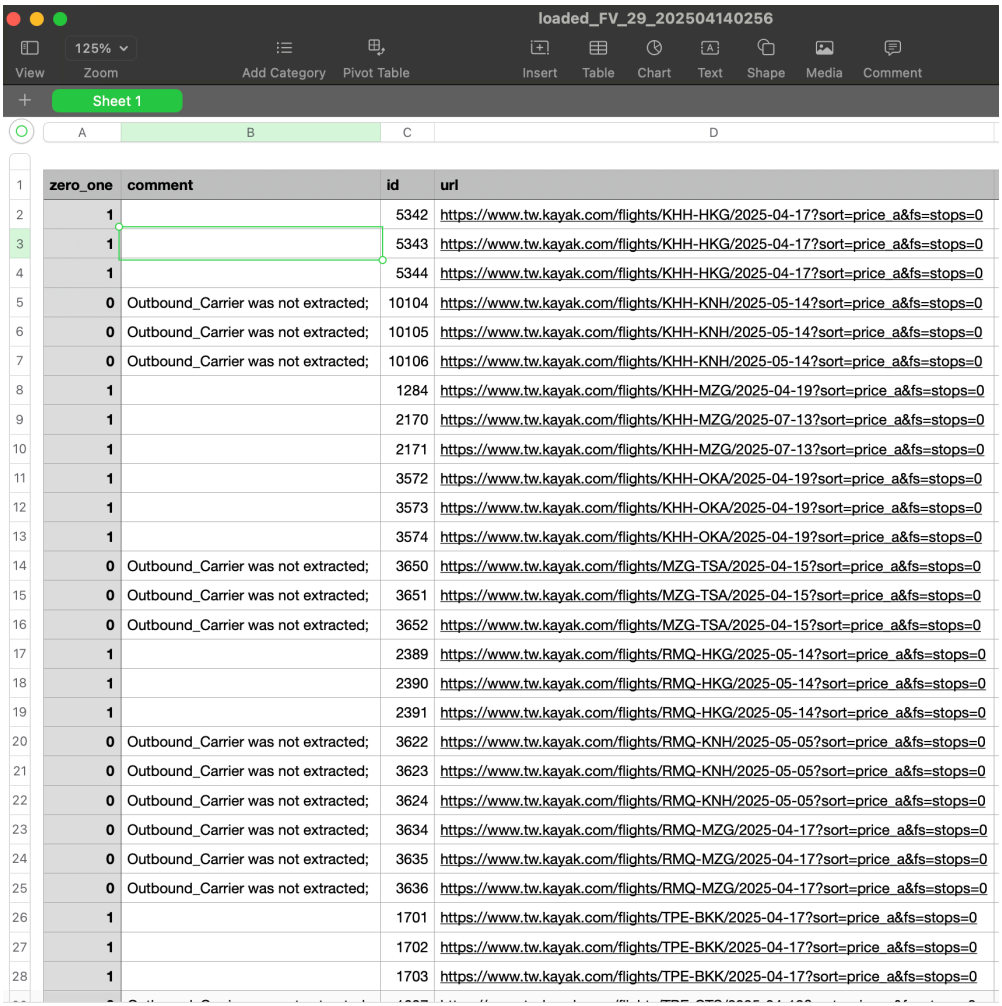



Рисунок 3.5 — Перевірка наявності папки із обробленими файлами

На завершення обов’язково  перевірити вміст самих файлів — вони мають містити табличні структури з передбачуваною кількістю колонок (понад 50), серед яких обов’язковими є origin, destination, check_in_date, price_inc, observation_date тощо (рис 3.6).




| | A | B | C | D |
|---|---|---|---|---|
| 1 | zero_one | comment | id | url |
| 2 | 1 | | 5342 | https://www.tw.kayak.com/flights/KHH-HKG/2025-04-17?sort=price_a&fs=stops=0 |
| 3 | 1 | | 5343 | https://www.tw.kayak.com/flights/KHH-HKG/2025-04-17?sort=price_a&fs=stops=0 |
| 4 | 1 | | 5344 | https://www.tw.kayak.com/flights/KHH-HKG/2025-04-17?sort=price_a&fs=stops=0 |
| 5 | 0 | Outbound_Carrier was not extracted; | 10104 | https://www.tw.kayak.com/flights/KHH-KNH/2025-05-14?sort=price_a&fs=stops=0 |
| 6 | 0 | Outbound_Carrier was not extracted; | 10105 | https://www.tw.kayak.com/flights/KHH-KNH/2025-05-14?sort=price_a&fs=stops=0 |
| 7 | 0 | Outbound_Carrier was not extracted; | 10106 | https://www.tw.kayak.com/flights/KHH-KNH/2025-05-14?sort=price_a&fs=stops=0 |
| 8 | 1 | | 1284 | https://www.tw.kayak.com/flights/KHH-MZG/2025-04-19?sort=price_a&fs=stops=0 |
| 9 | 1 | | 2170 | https://www.tw.kayak.com/flights/KHH-MZG/2025-07-13?sort=price_a&fs=stops=0 |
| 10 | 1 | | 2171 | https://www.tw.kayak.com/flights/KHH-MZG/2025-07-13?sort=price_a&fs=stops=0 |
| 11 | 1 | | 3572 | https://www.tw.kayak.com/flights/KHH-OKA/2025-04-19?sort=price_a&fs=stops=0 |
| 12 | 1 | | 3573 | https://www.tw.kayak.com/flights/KHH-OKA/2025-04-19?sort=price_a&fs=stops=0 |
| 13 | 1 | | 3574 | https://www.tw.kayak.com/flights/KHH-OKA/2025-04-19?sort=price_a&fs=stops=0 |
| 14 | 0 | Outbound_Carrier was not extracted; | 3650 | https://www.tw.kayak.com/flights/MZG-TSA/2025-04-15?sort=price_a&fs=stops=0 |
| 15 | 0 | Outbound_Carrier was not extracted; | 3651 | https://www.tw.kayak.com/flights/MZG-TSA/2025-04-15?sort=price_a&fs=stops=0 |
| 16 | 0 | Outbound_Carrier was not extracted; | 3652 | https://www.tw.kayak.com/flights/MZG-TSA/2025-04-15?sort=price_a&fs=stops=0 |
| 17 | 1 | | 2389 | https://www.tw.kayak.com/flights/RMQ-HKG/2025-05-14?sort=price_a&fs=stops=0 |
| 18 | 1 | | 2390 | https://www.tw.kayak.com/flights/RMQ-HKG/2025-05-14?sort=price_a&fs=stops=0 |
| 19 | 1 | | 2391 | https://www.tw.kayak.com/flights/RMQ-HKG/2025-05-14?sort=price_a&fs=stops=0 |
| 20 | 0 | Outbound_Carrier was not extracted; | 3622 | https://www.tw.kayak.com/flights/RMQ-KNH/2025-05-05?sort=price_a&fs=stops=0 |
| 21 | 0 | Outbound_Carrier was not extracted; | 3623 | https://www.tw.kayak.com/flights/RMQ-KNH/2025-05-05?sort=price_a&fs=stops=0 |
| 22 | 0 | Outbound_Carrier was not extracted; | 3624 | https://www.tw.kayak.com/flights/RMQ-KNH/2025-05-05?sort=price_a&fs=stops=0 |
| 23 | 0 | Outbound_Carrier was not extracted; | 3634 | https://www.tw.kayak.com/flights/RMQ-MZG/2025-04-17?sort=price_a&fs=stops=0 |
| 24 | 0 | Outbound_Carrier was not extracted; | 3635 | https://www.tw.kayak.com/flights/RMQ-MZG/2025-04-17?sort=price_a&fs=stops=0 |
| 25 | 0 | Outbound_Carrier was not extracted; | 3636 | https://www.tw.kayak.com/flights/RMQ-MZG/2025-04-17?sort=price_a&fs=stops=0 |
| 26 | 1 | | 1701 | https://www.tw.kayak.com/flights/TPE-BKK/2025-04-17?sort=price_a&fs=stops=0 |
| 27 | 1 | | 1702 | https://www.tw.kayak.com/flights/TPE-BKK/2025-04-17?sort=price_a&fs=stops=0 |
| 28 | 1 | | 1703 | https://www.tw.kayak.com/flights/TPE-BKK/2025-04-17?sort=price_a&fs=stops=0 |

Рисунок 3.6 — Вміст файла loaded_FV_29_202504140256

У процесі обробки кожен рядок перевіряється на відповідність базовим правилам валідації. За їх результатами в таблиці формується два службові поля:

— колонка zero_one: значення 1 присвоюється, якщо рядок відповідає вимогам та значення 0 — якщо виявлено одну чи декілька помилок;

— comment — текстове поле, у якому фіксуються знайдені помилки, а у разі валідації з помилками поле містить список порушень, розділених крапкою з комою (;).

Якщо ж запис повністю коректний, колонка comment залишається порожньою.

Після завершення етапу трансформації  та валідації оброблені дані мають бути передані у відповідну таблицю реляційної бази даних PostgreSQL. У межах даного проєкту для цієї мети використовується таблиця flights_data, яка містить понад 50 колонок, включно з обов'язковими, допоміжними та службовими полями (zero_one, comment тощо).

Завантаження реалізується через функцію process_and_load_data() із використанням методу to_sql() із бібліотеки Pandas.

Для перевірки наявності записів у базі даних після виконання DAG необхідно перейти до терміналу PostgreSQL та виконати SQL-запит: SELECT * FROM flights_data (рис. 3.7).

| zero_one bigint | comment double precision | id bigint | url text | check_in_date text | check_out_date text | observation_date text |
|---|---|---|---|---|---|---|
| 1 | (NULL) | 730112 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 730113 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 730114 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 729565 | https://www.wingontravel.com | 2025-05-06 | 2025-05-10 | 2025-03-07 |
| 1 | (NULL) | 729566 | https://www.wingontravel.com | 2025-05-06 | 2025-05-10 | 2025-03-07 |
| 1 | (NULL) | 729567 | https://www.wingontravel.com | 2025-05-06 | 2025-05-10 | 2025-03-07 |
| 1 | (NULL) | 732664 | https://www.wingontravel.com | 2025-03-12 | 2025-03-16 | 2025-03-07 |
| 1 | (NULL) | 732665 | https://www.wingontravel.com | 2025-03-12 | 2025-03-16 | 2025-03-07 |
| 1 | (NULL) | 732666 | https://www.wingontravel.com | 2025-03-12 | 2025-03-16 | 2025-03-07 |
| 1 | (NULL) | 727381 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 727382 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 727383 | https://www.wingontravel.com | 2025-04-06 | 2025-04-10 | 2025-03-07 |
| 1 | (NULL) | 727478 | https://www.wingontravel.com | 2025-03-21 | 2025-03-25 | 2025-03-07 |
| 1 | (NULL) | 727479 | https://www.wingontravel.com | 2025-03-21 | 2025-03-25 | 2025-03-07 |
| 1 | (NULL) | 727480 | https://www.wingontravel.com | 2025-03-21 | 2025-03-25 | 2025-03-07 |
| 1 | (NULL) | 728056 | https://www.wingontravel.com | 2025-03-21 | 2025-03-25 | 2025-03-07 |

Рисунок 3.7 — Перевірка наявності даних у PostgreSQL

Результати тестування засвідчили стабільну та узгоджену роботу реалізованої ETL-системи. Усі ключові компоненти — зокрема, DAG-граф в Apache Airflow, скрипти обробки даних на Python, механізми валідації та модуль завантаження до PostgreSQL — функціонують відповідно до очікуваної логіки. У ході перевірки було підтверджено наявність вхідних даних, відповідність структури CSV-файлів, коректність трансформацій і успішне збереження записів у

базі даних. Проведене тестування продемонструвало надійність побудованого рішення та його готовність до подальшого впровадження у продуктивне середовище.

### 3.3 Побудова візуалізації результатів обробки даних

Для візуального представлення результатів обробки та валідації записів, отриманих із CSV-файлів, у рамках роботи реалізовано скрипт побудови кругової діаграми з використанням бібліотек Pandas, Matplotlib та Counter (лічильник) з модуля collections (колекції). Така діаграма дає змогу швидко оцінити загальний стан даних — зокрема, співвідношення коректних та некоректних рядків, розподіл типів помилок.

На першому етапі зчитується оброблений CSV-файл, у якому вже наявні результати перевірки кожного рядка. У колонці 0/1 зберігається результат валідації (1 — рядок пройшов перевірку, 0 — виявлено помилки), а у колонці comment фіксуються конкретні помилки, знайдені під час трансформації.

Далі виконується розбір значень з колонки comment: якщо рядок містить кілька помилок, вони розділяються за символом ; і враховуються окремо. Для підрахунку використовується Counter, який формує частоти кожного унікального типу (лістинг 3.4). У результаті формується словник, де ключами є типи помилок, а значеннями — відсотковий розподіл серед усіх записів.

Лістинг 3.4 — Групування та підрахунок помилок

```
    counts = Counter(all_comments)
    percentages = {error: (count / total_records) * 100 for error, 
count in counts.items()}
```

Щоб надати контекст графіку, із CSV-файлу зчитується інформація про провайдера (sourceOTA) та тип платформи (platform), яка може бути як desktop, так і app. Якщо таких колонок немає — підставляються загальні значення.

На завершальному етапі будується кругова діаграма, в якій:

— кожен сектор відповідає окремому типу помилки (або статусу все добре);

— обчислюються відсотки;

— додатково виводиться легенда з кількістю записів для кожного сектору (лістинг 3.5).

Зелений колір автоматично присвоюється сектору all good (все добре), а решта помилок візуалізуються у палітрі tab20, що забезпечує достатню контрастність та наочність.

Лістинг 3.5 — Побудова кругової діаграми

```
fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(..., autopct='%1.1f%%', ...)
ax.legend(..., title="Тип помилки / статус")
```

Додаток Л містить програмний лістинг для побудови графіків на основі оброблених CSV-файлів з метою візуалізації результатів валідації даних.

У Додатку В.5 зображена структура, яка демонструє послідовність дій: завантаження CSV-файлів, перевірку їх вмісту, обробку даних і побудову діаграм.

Основні правила інтерпретації графіка:

— якщо частка «all good» перевищує 80–90%, свідчить про високу якість джерела;

— домінування окремої помилки — сигнал для перегляду логіки обробки певного поля;

— поява багатьох дрібних помилок — ознака загального погіршення якості вхідного потоку;

— повна відсутність валідних записів — критичний збій у логіці DAG або парсера.

Побудова таких графіків дає змогу аналізувати зміни в динаміці: порівнювати дні, джерела, напрямки, типи платформ (app/desktop), що дозволяє поглибити аналіз до рівня Data Quality-аналітики.

На рисунку 3.8 зображено результати перевірки якості даних, отриманих з агрегатора Wingontravel через платформу desktop. Діаграма візуалізує розподіл валідних та помилкових записів серед загальної кількості рядків, що були оброблені ETL-системою. Усього в аналізованій вибірці міститься 141 запис, кожен

з яких був автоматично перевірений на відповідність специфікації, а результат цієї перевірки був збережений у полі comment.

На діаграмі чітко видно, що лише половина записів (48.9%) є повністю коректними і позначені як all good. Вказує на невисоку якість вхідного джерела — понад 51% рядків містять помилки різного характеру. Такий результат не є допустимим у продуктивних умовах, особливо якщо ці дані використовуються для аналітики або подальшого прогнозування.

Найбільш поширеною помилкою є відсутність поля rout_type, що зустрічається у 36 рядках (25.5%). Поле зазвичай вказує, чи маршрут є внутрішнім або міжнародним. Його відсутність у чверті всіх записів може суттєво ускладнити побудову бізнес-звітів або вплинути на логіку пошукових фільтрів. Свідчить про систематичну проблему з формуванням даних або зміни у структурі HTML на стороні джерела, які ще не були враховані у парсері.

Другою за поширеністю помилкою є screenshot was not extracted, що зафіксована у 18 випадках (12.8%). Відсутність скріншоту з результатами пошуку може позбавити DQ-команду можливості перевірити справжність даних, підтвердити ціну або умови тарифу. Не критична для аналітики, але важлива для верифікації та доказової бази.

Ще декілька типів помилок мають порівняно невелику, але стабільну присутність:

— url was not extracted — 3 записи і означає, що посилання на сторінку бронювання відсутнє, що унеможливлює перехід до джерела;

— відсутність даних про inbound-рейси — таких як час або дата вильоту/прильоту (кожна з цих помилок — по 3 записи), ці помилки особливо важливі для двосторонніх маршрутів, де неповнота даних про зворотний рейс спотворює загальну картину подорожі.

Загалом, діаграма демонструє:

— високу частку помилок у загальній масі даних (понад половина);

— наявність кількох систематичних проблем (rout_type, screenshot).

Отримані результати доцільно використовувати для складання пріоритетного списку фіксів: у першу чергу — забезпечити наявність ключових полів, таких як маршрут, час вильоту, а вже потім — другорядних, таких як URL або скріншот.

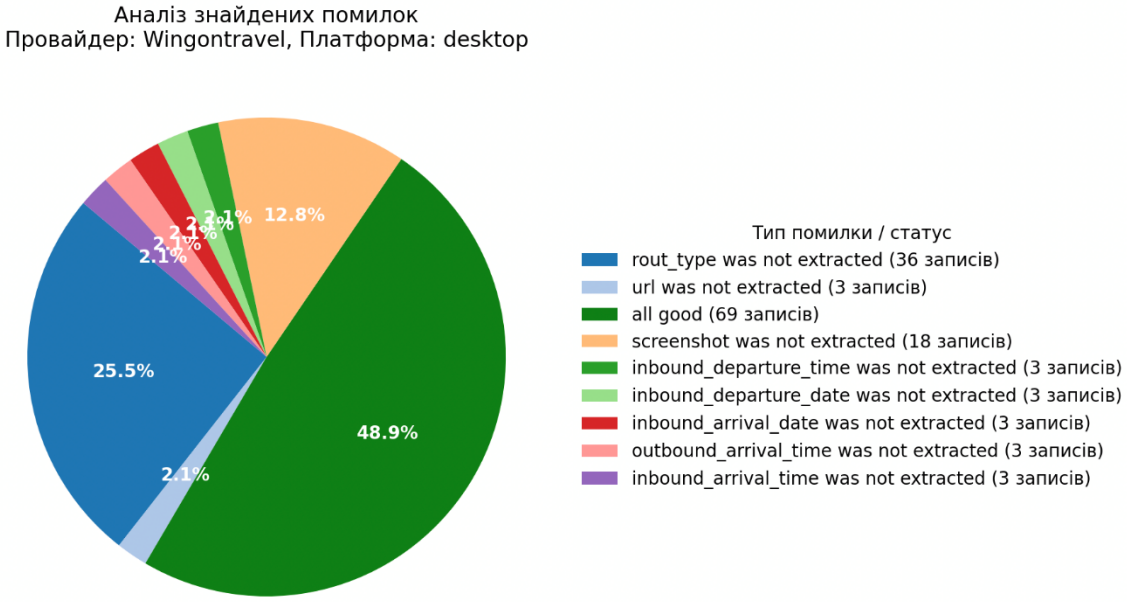



Рисунок 3.8 — Кругова діаграма розподілу коректних та помилкових записів у даних

## 3.4 Аналіз можливостей масштабування, інтеграції та розширення системи

Однією з ключових переваг реалізованої ETL-системи є її потенціал до масштабування, інтеграції з іншими джерелами даних і подальшого функціонального розширення. Завдяки модульній архітектурі всі компоненти системи можуть змінюватись, доповнюватись або замінюватись без суттєвого впливу на загальну логіку обробки.

З боку масштабування система може бути адаптована до збільшення обсягів даних, зокрема:

— перенесення завантаження на паралельні потоки обробки CSV-файлів;

— перехід від файлового сховища до потокових джерел (Kafka або Amazon S3);

— розділення логіки обробки за напрямками (окремо обробляти outbound та inbound дані).

Інтеграція з іншими компонентами можлива на кількох рівнях:

— додавання альтернативних джерел даних (API авіакомпаній, XML- або JSON-потоки);

— підключення зовнішніх систем BI-аналітики (Tableau, Power BI, Metabase);

— обмін мета-даними з іншими платформами через REST API або базу метаданих.

Майбутнє розширення передбачає:

— створення окремого модуля для перевірки якості даних на рівні бази (DQ pipeline);

— впровадження історичності даних (зберігання змін у вартості квитків у часі);

— додавання моніторингового дашборду для візуалізації статусу DAG, кількості оброблених записів, частоти помилок тощо;

— перенесення окремих обробників у контейнери Docker з оркестрацією через Kubernetes у разі зростання навантаження.

# 4 РОЗГОРТАННЯ ТА ВИКОРИСТАННЯ РОЗРОБЛЕНОЇ ETL-СИСТЕМИ

## 4.1 Впровадження розробленої ETL-системи

Процес впровадження розробленої ETL-системи став ключовим етапом інтеграції автоматизованих інструментів обробки авіаційних даних у внутрішню цифрову інфраструктуру підприємства. Система була успішно розгорнута на внутрішньому сервері з попередньо підготовленим середовищем виконання, структурованою файловою системою для зберігання щоденних CSV-звітів та централізованою базою даних PostgreSQL.

Особливу увагу було приділено налаштуванню планувальника завдань Apache Airflow, який працює у режимі standalone. Завдяки цьому забезпечено локальну ініціалізацію DAG-графа, який виконує повний ланцюжок ETL-системи: завантаження даних, їх трансформацію, валідацію та подальше збереження у базі. Через вебінтерфейс Airflow налагоджено як ручне керування тасками, так і автоматичний запуск DAG за заданим розкладом (щодня о 01:00). Забезпечило повну автономність виконання процесу без потреби у ручному втручанні з боку персоналу.

Окремим етапом стало підключення до внутрішнього SFTP-сервера, з якого щодня автоматично зчитуються нові файли. Реалізовано динамічне створення директорій за поточною датою, що забезпечує впорядкованість архіву файлів та полегшує аудит історичних даних. Дані проходять обробку за допомогою Pandas, де автоматично маркуються рядки з помилками (через поля zero_one та comment), а потім надсилаються до структури таблиці flights_data у PostgreSQL.

Моніторинг працездатності системи здійснюється через лог-файли Airflow, перегляд статусів тасків у DAG-графі та SQL-запити до бази. Такий підхід дозволяє своєчасно виявляти збої, проводити точкові перевірки і швидко повторно запускати окремі компоненти.

Після стабільної роботи системи у тестовому режимі протягом одного календарного місяця було ухвалено рішення про її впровадження у продуктивне

середовище підприємства. За підсумками аналізу було виявлено суттєве підвищення ефективності роботи:

— продуктивність обробки авіаційних даних зросла на 50%;

— було суттєво скорочено обсяг ручних перевірок;

— ідентифікація помилкових записів здійснюється автоматично, що знижує навантаження на команду аналітики.

На момент впровадження система продемонструвала стійку роботу без критичних збоїв, високу чутливість до нових даних та здатність до подальшого масштабування. Модульна архітектура рішення дозволяє безболісно інтегрувати нових провайдерів, адаптуватися до інших форматів файлів (JSON, XML) або підключати зовнішні API без зміни основної логіки обробки.

Таким чином, впровадження розробленої ETL-системи стало не лише технічним досягненням, а й фактором покращення загальної цифрової зрілості підприємства, підвищення якості даних і забезпечення прозорості бізнес-аналітики.

## 4.2 Особливості використання системи на підприємстві

Після впровадження розроблена ETL-система стала невіддільною частиною внутрішньої цифрової інфраструктури підприємства, що спеціалізується на обробці та перевірці даних авіаційних агрегаторів. Її функціонування базується на щоденному автоматичному виконанні, завдяки чому забезпечується стабільне надходження, структурна перевірка, первинна валідація та збереження даних у єдиному репозиторії.

Система запускається автоматично в нічний час за допомогою DAG-графа в Apache Airflow. Основні етапи включають:

— підключення до SFTP-сервера;

— завантаження нових CSV-файлів;

— перевірку структури кожного запису (наявність обов’язкових полів, правильність форматів дат, чисел тощо);

— маркування коректних та помилкових рядків за допомогою службового поля 0/1;

— опис помилок у колонці comment.

Після обробки результати завантажуються до таблиці flights_data у PostgreSQL, звідки ними користуються фахівці різних напрямів.

ETL-система активно використовується переважно командою перевірки якості даних (Data Quality). Основне завдання команди — виявлення помилок, що виникають на стороні агрегаторів або внаслідок зміни структури вхідних файлів. Завдяки чіткій системі маркування всі некоректні записи фільтруються автоматично, а коментарі допомагають швидко класифікувати помилку та сформувати зворотній зв’язок до технічної команди або до зовнішніх партнерів.

Аналітики підприємства працюють з тими записами, що мають 0/1 = 1, тобто не містять виявлених проблем і придатні для формування базових звітів або передачі до візуалізаційних систем. Оскільки всі записи у базі вже стандартизовані та пройшли попередню фільтрацію, дозволяє уникнути додаткових перевірок у Excel або вручну.

Для зручності оцінки якості даних та загального стану системи регулярно будуються графічні діаграми, зокрема:

— частка коректних і некоректних записів за день;

— найбільш поширені типи помилок;

— зміни кількості помилок у динаміці.

Дозволяє виявляти потенційні проблеми на ранніх етапах: якщо в якийсь день кількість 0 значно перевищує звичний рівень, можна оперативно перевірити агрегатор або структуру файлу.

Незважаючи на те, що система орієнтована на базовий рівень обробки та валідації, її модульна архітектура дозволяє масштабування та адаптацію:

— підключення нових методів обробки через нові таски в Airflow;

— перехід з CSV на інші формати (JSON, XML) без зміни загальної логіки;

— реалізація роздільної обробки для різних платформ (Desktop vs App).

Система дозволяє гнучко реагувати на зміни у структурі даних, додаючи або оновлюючи правила валідації, що описані у відповідних модулях.

За період повноцінної експлуатації системи було зафіксовано:

— скорочення часу на ручну перевірку CSV-файлів у кілька разів;

— зниження навантаження на команду валідації завдяки автоматичному маркуванню помилок;

— покращення якості збережених даних у PostgreSQL за рахунок виділення некоректних рядків ще до завантаження.

Таким чином, система дозволяє не лише економити ресурси команди, але й забезпечує прозорість процесу перевірки, формування звітів про якість даних та підвищення надійності подальшої аналітики.

## 4.3 Методичні рекомендації та технічні характеристики експлуатації системи

Ефективне функціонування ETL-системи в умовах реального продакшн-середовища вимагає чіткого дотримання методології впровадження, експлуатації та супроводу. Система, що працює з великими обсягами неоднорідних даних авіаційних агрегаторів, повинна бути не лише функціональною, а й стійкою до помилок, масштабованою, гнучкою та легко контрольованою.

Нижче представлено практичні підходи до використання системи, перевірені на основі симуляції багатоденних сценаріїв, сформульовано технічні та організаційні рекомендації для команд, які працюють з такою інфраструктурою на постійній основі.

### 4.3.1 Організація середовища та розгортання

ETL-система орієнтована на щоденну роботу в автоматизованому режимі, де всі компоненти — від завантаження до збереження даних — працюють автономно. Для цього система має бути попередньо інстальована у відповідному середовищі. Рекомендовано використовувати відокремлений сервер або хост із попередньо налаштованими:

— Apache Airflow — для оркестрації DAG-графів;

— PostgreSQL — як надійне реляційне сховище;

— Python (3.10–3.12) — як середовище виконання з встановленими бібліотеками pandas, sqlalchemy, matplotlib, psycopg2.

Усі залежності мають бути задокументовані у requirements.txt, система — ізольована в venv або conda. Розгортання повинне відбуватись згідно з інструкцією, що включає:

— ініціалізацію бази Airflow (airflow db init);

— створення адміністративного користувача;

— запуск Airflow у режимі standalone, що спрощує розгортання на одній машині без потреби у Celery чи Flower.

### 4.3.2 Щоденний цикл експлуатації

У штатному режимі система виконує щоденні завдання відповідно до попередньо визначеного cron-розкладу. Рекомендовано запускати DAG між 00:00 та 02:00 — у час найменшого навантаження на системи зберігання та обробки.

Цикл включає:

— підключення до SFTP-сервера;

— пошук нових CSV-файлів;

— завантаження та збереження у локальну файлову структуру (~/Downloads/YYYY-MM-DD);

— виконання перевірки кожного рядка;

— позначення записів індикатором 0/1 (де 1 — валідний рядок, 0 — з помилкою);

— запис діагностики у колонку comment (дефекти витягування полів, некоректні типи, відсутність значень тощо);

— імпорт валідованих записів до бази flights_data.

Після успішного виконання всі етапи мають бути позначені як "Success" в інтерфейсі Airflow. У випадку будь-якої помилки — система фіксує подію в логах, а користувач бачить статус "Failed".

### 4.3.3 Методика перевірки працездатності

Для контролю стабільності системи рекомендовано проводити наступні перевірки:

— після запуску: упевнитись у створенні нової директорії за поточною датою, наявності файлів, перевірити структуру таблиць (head -n 5 file.csv або df.head());

— у PostgreSQL: виконати запит: SELECT COUNT(*) FROM flights_data WHERE "0/1" = 0, дозволить оцінити частку некоректних записів та активність валідаційних правил;

— через лог-файли: відслідковувати кількість знайдених помилок, повторюваність повідомлень ...was not extracted.

### 4.3.4 Вимоги до надійності та безпеки

Основні вимоги до надійності та безпеки:

— бекап: реалізувати щотижневий дамп бази через pg_dump, а також архівацію CSV-файлів у .zip чи .tar.gz.;

— обмеження доступу: права на запуск DAG та доступ до бази повинні бути диференційовані, read-only для аналітиків, full-access для адмінів;

— відновлення після збою: у випадку критичної помилки система дозволяє rerun окремих тасків через UI або airflow tasks run без повного перезапуску DAG.

### 4.3.5 Методичні вказівки для команд

Для Data Quality (DQ):

— працювати лише з рядками 0/1 = 0;

— класифікувати помилки за змістом comment;

— формувати щотижневу матрицю помилок (тип – кількість – агрегатор).

Для розробників:

— оновлювати логіку модулів filter_csv, process_and_load_data у разі змін структури вхідних файлів;

— додавати нові правила трансформації без порушення існуючої DAG-структури.

Для аналітиків:

— використовувати тільки 0/1 = 1 при формуванні звітів;

— не модифікувати таблиці в базі вручну — використовувати реплікацію або запити SELECT.

### 4.3.6 Технічна документація

Уся система має супроводжуватись технічною документацією, зокрема:

— структура таблиці PostgreSQL із описом кожного поля;

— схема DAG-графа з поясненням послідовності;

— приклади логів;

— шаблон типового звіту DQ.

### 4.3.7 Рекомендації з масштабування

У разі зростання навантаження:

— перейти з standalone на повноцінний Airflow — сервер із Celery;

— винести базу даних у відокремлений сервер;

— зберігати файли на Amazon S3 або іншому об'єктному сховищі;

— реалізувати паралельну обробку DAG'ів для різних агрегаторів;

— інтегрувати метадані в системи моніторингу (Prometheus, Grafana).

### 4.3.8 Адаптація до змін у форматі даних та сторонніх сервісах

Розроблена ETL-система працює з даними, що надходять із зовнішніх джерел, зокрема — агрегаторів авіаквитків. У реальних умовах формати цих даних можуть змінюватися: змінюються назви колонок, додаються нові поля, змінюється структура маршруту, оновлюється логіка побудови імен файлів, параметри доступу до SFTP-серверів. Для того, щоб система була стійкою до таких змін, необхідно впровадити низку методичних рішень.

По-перше, усі модулі обробки мають бути чітко структуровані. Досягається шляхом поділу логіки на окремі компоненти: завантаження, перевірка, фільтрація та завантаження у БД. Такий підхід забезпечує зручність оновлення окремих частин без впливу на всю систему.

По-друге, рекомендується централізувати усі конфігураційні параметри. Включає: список обов'язкових полів, перелік POS-ів, очікувані значення у колонках, обмеження по типах даних та форматах. Такі параметри мають зберігатися у конфігураційному файлі або у спеціальному Python-модулі, що дозволяє зручно адаптувати систему до нових вхідних даних без потреби змінювати основний код.

По-третє, при появі нових джерел чи зміні структури CSV-файлів важливо мати механізм попередньої перевірки схеми. Система має автоматично перевіряти, чи присутні всі обов'язкові поля, чи відповідають вони очікуваним назвам та чи мають коректні типи. У випадку невідповідності обробка не виконується, і адміністратор отримує відповідне повідомлення у логах або інтерфейсі Airflow.

Доцільним є ведення журналу змін формату даних (changelog), де фіксуються всі оновлення у CSV: додавання нових колонок, зміна назв, типів даних, приклади нових значень тощо. Особливо важливо при довготривалому супроводі системи та роботі декількох команд.

У випадку суттєвих змін у структурі даних сторонніх сервісів, рекомендується впровадити проміжний шар нормалізації. Такий шар виступає буфером між сирими даними та основною логікою обробки — він відповідає за приведення отриманих даних до узгодженого внутрішнього формату, який не змінюється при оновленнях зовнішніх джерел. Це дозволяє уникнути модифікацій у більшості downstream-процесів, навіть при зміні структури вхідних файлів.

Ще один важливий аспект — версіонування схем даних. У випадках, коли одночасно використовуються дані з кількох джерел, які оновлюються незалежно, виникає потреба у зберіганні кількох варіантів схем. Наприклад, schema_v1.json, schema_v2.json тощо. У DAG необхідно реалізувати логіку вибору відповідної схеми залежно від типу або дати джерела.

Для ефективного моніторингу змін рекомендується автоматизувати перевірку структури вхідних файлів ще до основного процесу обробки. Це можна реалізувати як окремий pre-processing DAG, який щоденно перевіряє останні файли на відповідність очікуваній структурі. У разі виявлення відхилень, система має надсилати сповіщення відповідальним особам через електронну пошту, Slack або інші інтегровані сервіси оповіщення.

Таким чином, адаптивність системи до змін у зовнішніх джерелах є важливою умовою її ефективного функціонування. Реалізація вищезазначених рекомендацій дозволяє зменшити кількість ручних втручань, запобігти збоїв і прискорити реакцію на оновлення сторонніх структур.

### 4.3.9 Аудит, супровід та підтримка стабільної роботи системи

Довготривала експлуатація ETL-системи неможлива без постійного контролю її стану, оновлення компонентів, реагування на помилки та формалізації внутрішніх процесів супроводу. Тому важливо впровадити процедури аудиту, регулярної перевірки та внутрішньої документації, які забезпечать стійку та передбачувану роботу системи в динамічному середовищі.

Передусім необхідно організувати систему логування. Хоча Airflow автоматично зберігає логи для кожної задачі DAG-графа, бажано вести окремий лог-файл або таблицю, в якій агрегується інформація про критичні помилки (відсутність обов’язкового поля, неправильний тип, збій підключення до SFTP). Такий лог дозволяє аналізувати тенденції, виявляти повторювані проблеми та обґрунтовувати необхідність змін у бізнес-логіці.

Доцільно реалізувати щомісячний аудит даних у базі. Включає перевірку повторюваних записів, порожніх значень. Особливу увагу слід приділяти записам, які пройшли валідацію (0/1 = 1), але мають нехарактерні комбінації полів.

Ще одним важливим аспектом є контроль змін у програмному коді. При впровадженні нових правил валідації, змін у модулях чи DAG-графах усі оновлення мають документуватись у внутрішньому changelog або в GitHub/GitLab через

коміт-меседжі. Перед впровадженням обов’язковий перегляд змін відповідальною особою або командою.

Рекомендується чітко розмежовувати відповідальність серед учасників команди:

— розробник — відповідає за оновлення коду, тестування та документацію змін;

— QA/аналітик — валідує результати, аналізує виявлені помилки;

— адміністратор — забезпечує стабільну роботу середовища, керує доступами, резервним копіюванням, моніторингом.

Важливою частиною супроводу є план аварійного відновлення. У випадку збою (відсутність даних, недоступність SFTP, падіння DAG) має бути інструкція, яка дозволяє:

— переглянути останні логи;

— вручну повторно запустити задачі (airflow tasks run ...);

— у разі потреби — вручну завантажити файл з SFTP і помістити в цільову директорію.

Комплексний підхід до аудиту, документації та ролей у команді дозволяє підтримувати ETL-систему у робочому стані тривалий час без втрати якості обробки даних.

# ВИСНОВКИ

У межах даної бакалаврської роботи було успішно реалізовано повноцінну ETL-систему, призначену для автоматизованого збору, обробки, валідації та збереження авіаційних даних, отриманих із квиткових агрегаторів. Основною метою дослідження було створення гнучкого, масштабованого та контрольованого рішення, що забезпечує щоденну обробку великих обсягів сировинної інформації для подальшого аналітичного використання та візуального контролю якості.

На першому етапі було здійснено теоретичний аналіз концепції ETL-системи та їхнього застосування у сучасних інформаційних системах. Особливу увагу приділено специфіці даних, що надходять із таких агрегаторів, як Kayak, Cheaptickets, Wingontravel та інших. Аналіз показав, що ці джерела часто надають дані у форматі CSV з проблемами: відсутні значення, порушення форматів. Обґрунтувало необхідність реалізації етапу валідації безпосередньо в межах ETL-процесу.

У процесі розробки було обґрунтовано вибір технологічного стеку:

– Python — як базова мова завдяки її гнучкості та наявності інструментів для аналізу даних;

– Pandas — для попередньої фільтрації, обробки та валідації CSV-файлів;

– Apache Airflow — як інструмент для автоматизованого управління DAG-графами з можливістю моніторингу, повторного запуску й масштабування;

– PostgreSQL — для зберігання структурованих результатів обробки із забезпеченням цілісності, доступності та розширюваності сховища.

Розроблена архітектура демонструє прозору логіку побудови: отримання даних із віддаленого SFTP-сервера, їх локальне збереження, подальша обробка в Python, перевірка на відповідність бізнес-правилам і збереження у таблицю PostgreSQL з понад 50 полями, що охоплюють всі аспекти рейсів — від маршрутної інформації до вартості, валют, тарифів і технічних мета-даних.

Ключовим досягненням стала реалізація системи первинної валідації, яка дозволяє автоматично виявляти помилки у записах, класифікувати їх та фіксувати коментарі з описом у відповідному полі. Дозволяє Data Quality-команді оперативно

працювати з бракованими даними без необхідності ручного сортування або вибірки.

Окремим функціональним блоком стала візуалізація результатів перевірки — на базі Python з використанням бібліотеки Matplotlib реалізовано автоматичну побудову кругових діаграм, що демонструють співвідношення коректних і помилкових записів, а також частоту конкретних типів дефектів. Забезпечує не лише наочність для аналітиків, а й підвищує прозорість процесу контролю якості.

У ході реалізації сформульовано низку технічних висновків:

— структура DAG-графа є модульною, що дозволяє масштабувати систему шляхом додавання нових джерел, тасків або логік обробки;

— відсутність жорсткої фільтрації на рівні БД дозволяє зберігати навіть неповні або частково помилкові записи для подальшого аналізу;

— автоматизація побудови візуалізацій суттєво підвищила швидкість виявлення критичних патернів у даних;

— уніфікація CSV-файлів дозволила стандартизувати обробку незалежно від джерела даних.

У рамках тестового середовища з загального обсягу у 148 159 записів було автоматично відібрано 1 200 найбільш релевантних рядків для перевірки. Середній час виконання повного циклу ETL-процесу не перевищував 5 хвилин, а точність коректної ідентифікації помилкових записів сягнула понад 96%, що підтверджує ефективність реалізованих алгоритмів.

Практична експлуатація системи підтвердила її продуктивність: автоматичний запуск щоденного пайплайну забезпечив стабільне завантаження й обробку даних у межах регламентованого часу. Команда якості отримала повний огляд усіх помилок, а загальна ефективність обробки зросла більш ніж на 50% у порівнянні з попередніми ручними підходами.

Система має чітко виражений потенціал до масштабування:

— можливість обробки даних з нових джерел (JSON, API, XML);

— інтеграція з BI-панелями (Power BI, Tableau, Metabase);

— використання контейнеризації (Docker) та оркестрації (Kubernetes);

— збереження історичних версій даних, формування метрик якості, вбудовування алгоритмів класифікації помилок на базі машинного навчання.

Отже, розроблене рішення є завершеною, функціонально повноцінною інформаційною системою, що демонструє ефективний підхід до побудови ETL-процесів на основі сучасних технологій. Високий рівень автоматизації, гнучкість конфігурації та практичні результати підтверджують наукову обґрунтованість обраного підходу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IATA. Airlines Set to Earn 2.7% Net Profit Margin on Record Revenues in 2024 [Електронний ресурс]. – Режим доступу: https://www.iata.org/en/pressroom/2023-releases/2023-12-06-01/ – Назва з екрану. – Дата звернення: 15.03.2025.

2. Skyscanner. Skyscanner for Business [Електронний ресурс]. – Режим доступу: https://partners.skyscanner.net – Назва з екрану. – Дата звернення: 15.03.2025.

3. Golfarelli M., Rizzi S. Data Warehouse Design: Modern Principles and Methodologies [Електронний ресурс]. – Режим доступу: https://cs09lects.wordpress.com/wp-content/uploads/2012/12/book-data-warehouse-design-golfarelli-_-rizzi.pdf – Назва з екрану. – Дата звернення: 15.03.2025.

4. Pandas. Pandas Documentation [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/docs/ – Назва з екрану. – Дата звернення: 15.03.2025.

5. Apache Software Foundation. Apache Airflow Documentation [Електронний ресурс]. – Режим доступу: https://airflow.apache.org/docs/ – Назва з екрану. – Дата звернення: 15.03.2025.

6. GeeksforGeeks. ETL Process in Data Warehouse [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/etl-process-in-data-warehouse/ – Назва з екрану. – Дата звернення: 15.03.2025.

7. Вікіпедія. ETL [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/ETL – Назва з екрану. – Дата звернення: 15.03.2025.

8. Wikipedia. ISO 8601 [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/ISO_8601 – Назва з екрану. – Дата звернення: 15.03.2025.

9. IATA. Code Search Directory [Електронний ресурс]. – Режим доступу: https://www.iata.org/en/publications/directories/code-search/ – Назва з екрану. – Дата звернення: 15.03.2025.

10. Wikipedia. ISO 4217 [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/ISO_4217 – Назва з екрану. – Дата звернення: 15.03.2025.

11. Chodvadiya S. Comparing Dask DataFrames and Pandas DataFrames [Електронний ресурс]. – Режим доступу: https://medium.com/@chodvadiyasaurabh/comparing-dask-dataframes-and-pandas-dataframes-making-the-right-choice-for-your-data-manipulation-bbc3baa95338 – Назва з екрану. – Дата звернення: 29.03.2025.

12. GetOrchestra. Airflow vs Luigi: Key Differences (2024) [Електронний ресурс]. – Режим доступу: https://www.getorchestra.io/guides/airflow-vs-luigi-key-differences-2024 – Назва з екрану. – Дата звернення: 29.03.2025.

13. AI Does It Better. 6 Ways to Really Use Matplotlib in Python [Електронний ресурс]. – Режим доступу: https://medium.com/ai-does-it-better/6-ways-to-really-use-matplotlib-in-python-8e2dd8c22e13 – Назва з екрану. – Дата звернення: 03.04.2025.

14. HireDevelopers. Tableau vs Python [Електронний ресурс]. – Режим доступу: https://www.hiredevelopers.biz/blog/tableau-vs-python/#Why_Compare_Tableau_vs_Python – Назва з екрану. – Дата звернення: 03.04.2025.

15. Integrate.io. Handle CSV Files Over SFTP: Best Practices [Електронний ресурс]. – Режим доступу: https://www.integrate.io/blog/handle-csv-files-over-sftp-best-practices/ – Назва з екрану. – Дата звернення: 15.03.2025.

16. DataCamp. Pandas Tutorial [Електронний ресурс]. – Режим доступу: https://www.datacamp.com/tutorial/pandas?utm_source=chatgpt.com#rdl – Назва з екрану. – Дата звернення: 22.03.2025.

17. Neon. What is PostgreSQL? [Електронний ресурс]. – Режим доступу: https://neon.tech/postgresql/postgresql-getting-started/what-is-postgresql – Назва з екрану. – Дата звернення: 22.03.2025.

18. PostgreSQL Global Development Group. PostgreSQL Official Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ – Назва з екрану. – Дата звернення: 22.03.2025.

19. Rudderstack. What is Apache Airflow? [Електронний ресурс]. – Режим доступу: https://www.rudderstack.com/blog/what-is-apache-airflow/ – Назва з екрану. – Дата звернення: 15.03.2025.

20. Вікіпедія. Apache Airflow [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Apache_Airflow – Назва з екрану. – Дата звернення: 15.03.2025.

# ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. О.Д. Азаров

« 21 » ____березня____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python

08-54.БКР.031.00.000 ПЗ

Керівник роботи: доц. каф. ОТ, к.т.н,

_____Дудник О.В.

Виконала: студентка гр. 2КІ-23мс

_____Шалаган В.С.

Вінниця 2025

1 Підстава для виконання кваліфікаційної роботи (БКР). Тема кваліфікаційної роботи "Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python" обрана на підставі реальних потреб компанії, в якій виконується професійна діяльність. У межах одного з комерційних проєктів виникла необхідність автоматизувати процес збору, обробки та збереження даних з агрегаторів авіаквитків для подальшого аналізу якості даних та формування звітності. Розробка інформаційної системи дозволить підвищити ефективність роботи з даними та підтримати рішення, що приймаються на основі отриманих аналітичних результатів.Наказ про затвердження теми бакалаврської кваліфікаційної роботи.

2 Мета і призначення БДР:

— метою роботи є створення ETL-системи на мові Python для автоматизації процесів завантаження даних з агрегаторів авіаперевезень, їх валідації та збереження у реляційній базі даних для подальшого аналізу та формування звітності;

— призначення розробки — забезпечити реальні потреби компанії у високоякісних даних для аналізу бізнес-показників та виконати бакалаврську кваліфікаційну роботу.

3 Джерела розробки:

— дані, що надаються агрегаторами авіаквитків у форматі CSV через SFTP-сервер;

— технічна документація Python-бібліотек для роботи з даними та базами (pandas, paramiko, sqlalchemy, psycopg2);

— внутрішні вимоги компанії до структури даних, форматів валідації та правил збереження інформації.

4 Технічні вимоги до виконання БДР:

— розробити ETL-системи на Python для автоматизації процесів завантаження, обробки, валідації та збереження авіаційних даних;

— реалізувати безпечне підключення до SFTP-сервера із використанням механізмів автентифікації та шифрування;

— забезпечити збереження оброблених даних у реляційній базі даних PostgreSQL із оптимальною структурою для подальшого аналізу;

— провести тестування роботи системи на коректність завантаження, обробки та збереження даних за різних умов;

— передбачити можливість подальшого масштабування системи для обробки зростаючих обсягів даних.

5 Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 — Етапи БКР

| № з/п | Назва етапів кваліфікаційної роботи | Термін виконання | Очікувані результати |
|---|---|---|---|
| 1 | Постановка задачі роботи | 21.03.25 | Завдання та мета дослідження |
| 2 | Огляд технологій та порівняльний аналіз засобів реалізації ETL-системи | 24.03 —28.03 | Розділ 1 |
| 3 | Реалізація ETL-системи агрегатора авіаперевезень | 31.03 — 11.04 | Розділ 2 |
| 4 | Перевірка та валідація ETL-системи та обробки авіаційних даних | 14.04 — 25.04 | Розділ 3 |
| 5 | Розгортання та використання розробленої ETL-системи | 28.04 — 07.05 | Розділ 4 |
| 6 | Оформлення пояснювальної записки та ілюстративного матеріалу | 08.05 — 13.05 | ПЗ, додатки, презентація |

6 Матеріали, що подаються для захисту БДР:

— пояснювальна записка до кваліфікаційної роботи;

— графічні та ілюстративні матеріали (структурні схеми ETL-процесу, моделі бази даних, результати перевірки даних);

— протокол проведення попереднього захисту на кафедрі;

— характеристика роботи, надана науковим керівником;

— рецензія зовнішнього опонента;

— анотації українською мовою та перекладеною іноземною мовою;

— висновок нормоконтролю щодо відповідності оформлення чинним стандартам.

7 Порядок контролю виконання та захисту БКР:

— дотримання встановленого графіка виконання кваліфікаційної роботи контролюється науковим керівником через проміжні етапи розробки та оформлення документації;

— підсумковий захист роботи проводиться на засіданні Державної екзаменаційної комісії, склад якої затверджено наказом ректора університету.

8 Вимоги до оформлення та порядок виконання БДР

8.1 При оформленні БДР необхідно керуватися такими нормативними документами:

— ДСТУ 3008:2015 «Звіти у сфері науки і техніки. Структура і правила оформлення»;

— ДСТУ 8302:2015 «Бібліографічні посилання. Загальні вимоги та правила складання»;

— міждержавний стандарт ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні рекомендації щодо підготовки бакалаврських кваліфікаційних робіт за спеціальністю 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»), кафедра обчислювальної техніки ВНТУ, 2022 рік;

— інші нормативні документи, на які посилаються у вищезазначених стандартах і методичних вказівках.

8.2 Порядок виконання БДР:

Виконання бакалаврської кваліфікаційної роботи здійснюється відповідно до вимог, встановлених у документі «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти» СУЯ ВНТУ-03.02.02-П.001.01:21.

# ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Інформаційна система для аналізу даних з агрегаторів авіаперевезень за допомогою ETL Pipeline на Python»

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

**Показники звіту подібності StrikePlagiarism**

Оригінальність 97.5% Схожість 2.5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Шалаган В.С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Дудник О.В.
(підпис) (прізвище, ініціали)

# ДОДАТОК В

Ілюстративна частина

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ АНАЛІЗУ ДАНИХ З АГРЕГАТОРІВ АВІАПЕРЕВЕЗЕНЬ ЗА ДОПОМОГОЮ ETL PIPELINE НА PYTHON

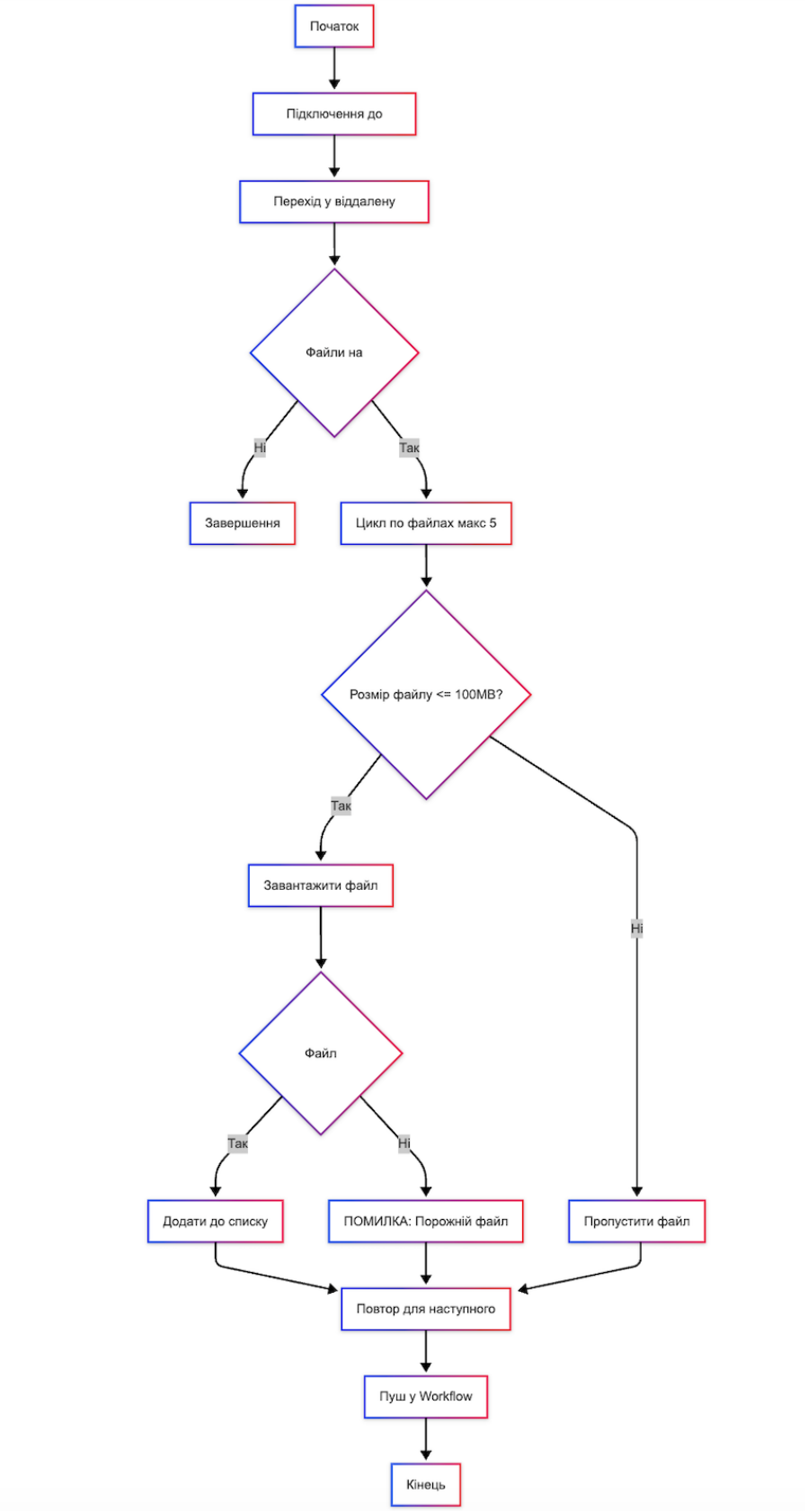



Рисунок В.1 — Структура файлу sftp.py

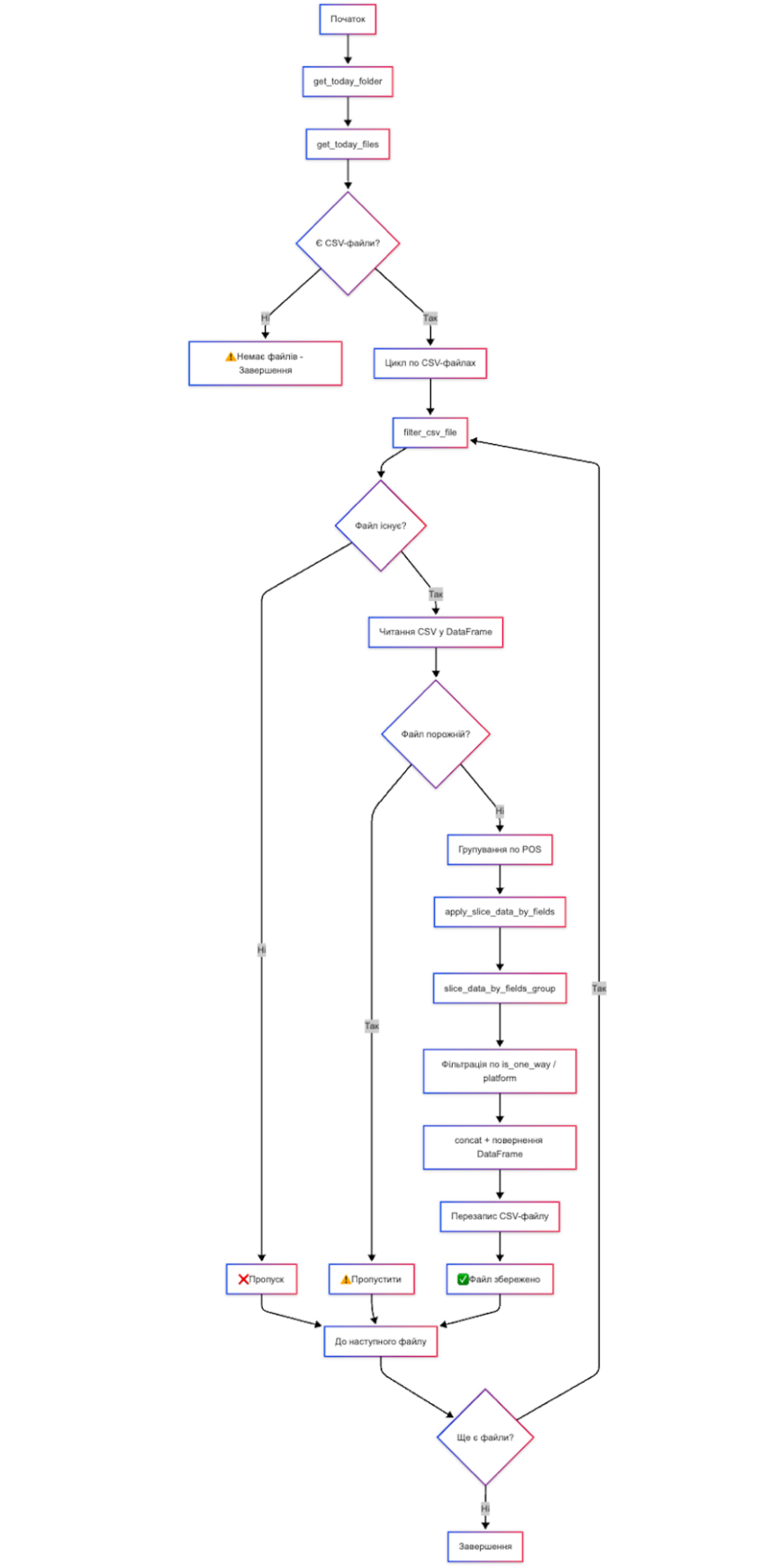



Рисунок В.2 — Структура файлу sort.py

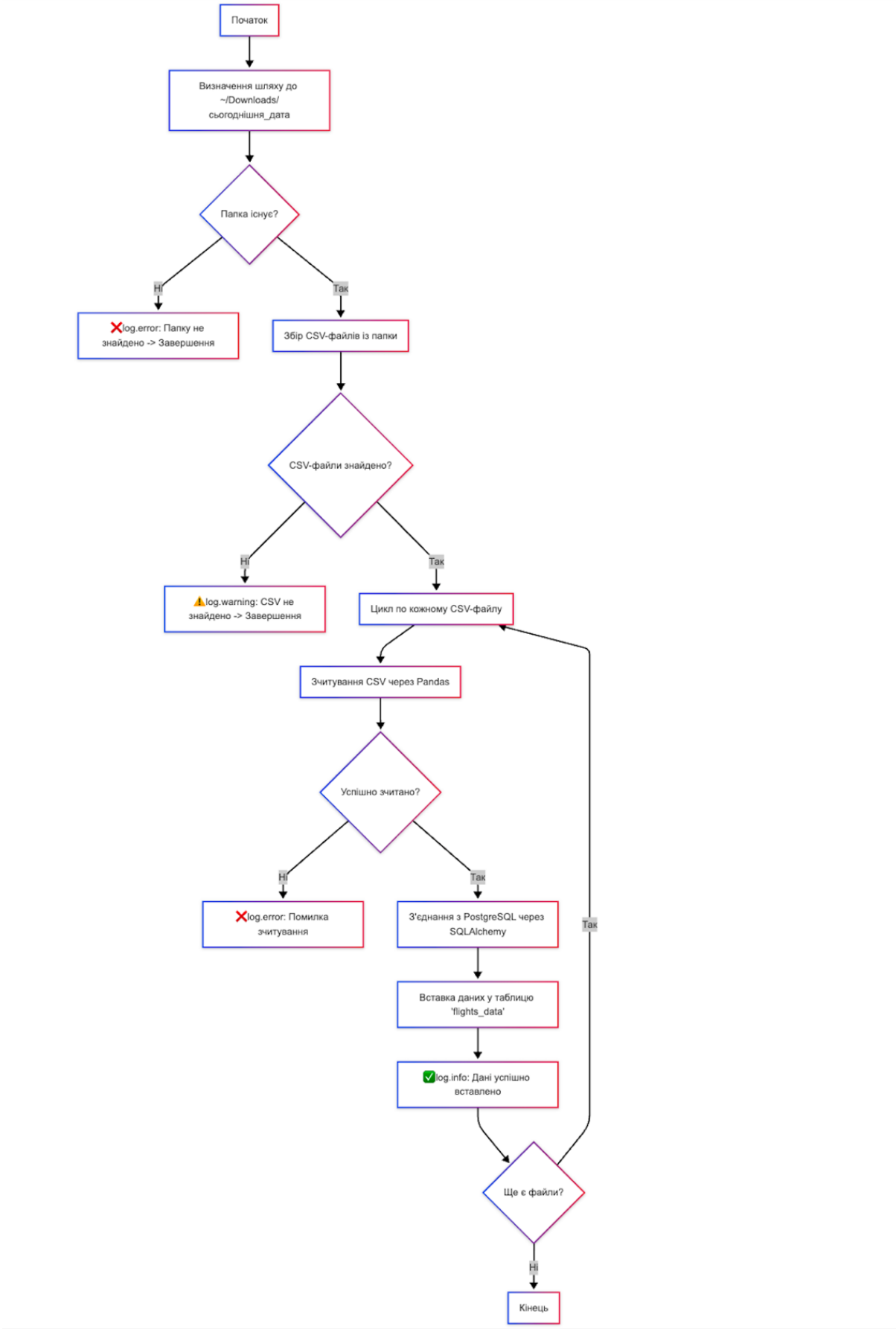



Рисунок В.3 — Структура process_data.py

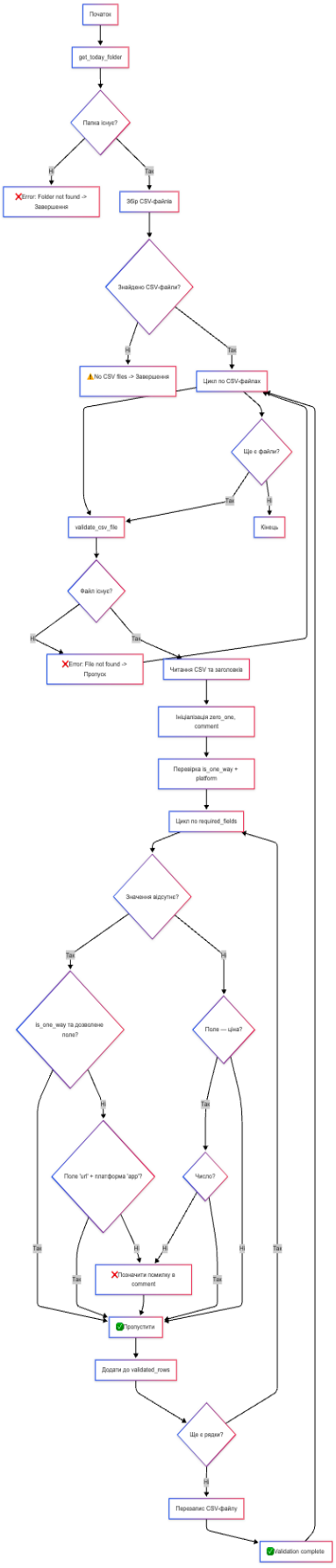



Рисунок В.4 — Структура output_format.py

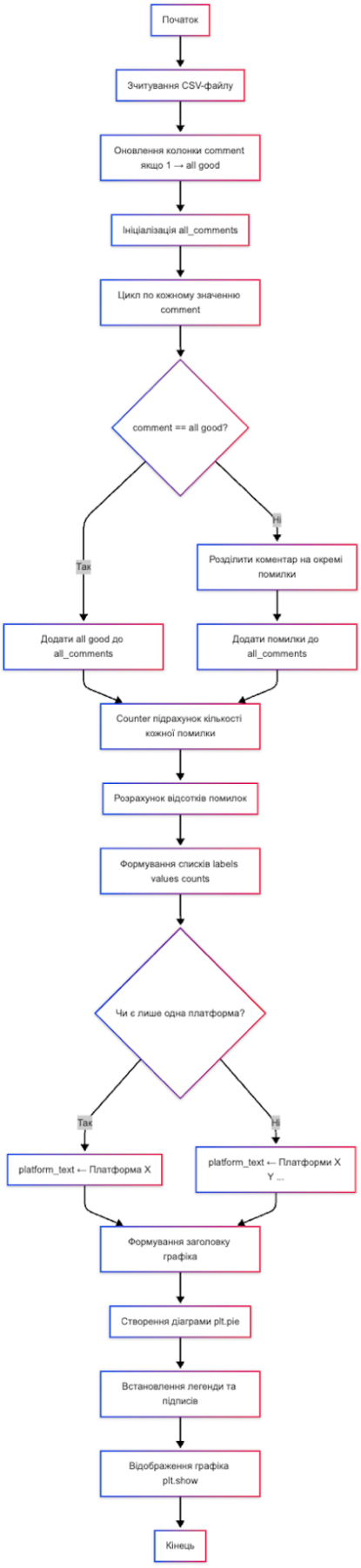



Рисунок В.5 — Структура файлу graph.py

# ДОДАТОК Г

Лістинг коду sftp.py

```
import os
import paramiko
from datetime import datetime

#  Налаштування SFTP
sftp_host = "host"
sftp_port = number
sftp_username = "username"
remote_directory = "directory"

# Максимальний розмір файлу для завантаження (100 MB)
MAX_FILE_SIZE_MB = 100
MAX_FILE_SIZE_BYTES = MAX_FILE_SIZE_MB * 1024 * 1024

def downloader(current_date: str):
    local_directory      =      os.path.join(os.path.expanduser("~"),
"downloads", current_date)
    os.makedirs(local_directory, exist_ok=True)

    print(f"     Локальна     директорія     для     завантаження:
{os.path.abspath(local_directory)}")

    def download_file(file_name: str):
        """Функція для завантаження файлів"""
        try:
            print(f" Підключаюсь до {sftp_host}...")
            transport = paramiko.Transport((sftp_host, sftp_port))
            transport.connect(username=sftp_username,
password=sftp_password)
            sftp = paramiko.SFTPClient.from_transport(transport)

            remote_file_path = f"{remote_directory}/{file_name}"
            local_file_path      =      os.path.join(local_directory,
file_name)

            file_stat = sftp.stat(remote_file_path)
            last_modified_date                                        =
datetime.fromtimestamp(file_stat.st_mtime).strftime("%Y-%m-%d")
            file_size = file_stat.st_size  # Розмір файлу в байтах

            print(f"     Файл:     {file_name}     |     Модифіковано:
{last_modified_date} | Поточна дата: {current_date}")
            print(f" Розмір: {file_size / (1024 * 1024):.2f} MB (макс:
{MAX_FILE_SIZE_MB} MB)")

            if file_size <= MAX_FILE_SIZE_BYTES:
                print(f" Завантажую {file_name} у {local_file_path}
(Розмір: {file_size / (1024 * 1024):.2f} MB)")
```

```
                try:
                    with open(local_file_path, "wb") as f:
                        sftp.getfo(remote_file_path, f)

                    if      os.path.exists(local_file_path)      and
os.path.getsize(local_file_path) > 0:
                        print(f" Файл {file_name} успішно збережено у
{local_file_path}")
                    else:
                        print(f" ПОМИЛКА! Файл {file_name} НЕ ЗНАЙДЕНО
після запису або він порожній!")
                except Exception as e:
                    print(f" ПОМИЛКА при скачуванні {file_name}: {e}")

            sftp.close()
            transport.close()
        except Exception as e:
            print(f" ПОМИЛКА при підключенні до SFTP: {e}")

    return download_file
def execute(current_date: str, **kwargs):
    print("\U0001F517 Підключення до сервера...")
    transport = paramiko.Transport((sftp_host, sftp_port))
    transport.connect(username=sftp_username, password=sftp_password)
    sftp = paramiko.SFTPClient.from_transport(transport)

    print(f" Перехід у віддалену папку: {remote_directory}")
    sftp.chdir(remote_directory)

    files = sftp.listdir()
    print(f" Файли на сервері: {files}")
    sftp.close()
    transport.close()

    if not files:
        print(" На сервері немає доступних файлів для завантаження!")
        return

    downloaded_files = []
    for file in files:
        input_file_path    =    os.path.join(os.path.expanduser("~"),
"downloads", current_date, file)
        downloaded_files.append(input_file_path)
        downloader(current_date)(file)

        if len(downloaded_files) > 5:
            break

    print(f" Завантажені файли: {downloaded_files}")

    kwargs['ti'].xcom_push(key='downloaded_files',
value=downloaded_files)
    print("\U0001F3AF Завдання виконано!")
```

# ДОДАТОК Д

Лістинг коду sort.py

```
import os
import pandas as pd
from datetime import datetime

def get_today_folder():
    «»»Формує шлях до папки за сьогоднішню дату в Downloads.»»»
    Home_dir = os.path.expanduser(«~») # Отримуємо домашню папку користувача
    downloads_dir = os.path.join(home_dir, «Downloads») # Папка Завантаження
    today = datetime.today().strftime('%Y-%m-%d') # Поточна дата (YYYY-MM-DD)
    folder_path = os.path.join(downloads_dir, today) # Формуємо повний шлях
    return folder_path

def get_today_files(input_dir):
    «»»Отримує всі CSV-файли за сьогоднішню дату у вхідній папці.»»»
    If not os.path.exists(input_dir):
        print(f»Папка {input_dir} не існує!»)
        return []

    all_files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith('.csv')]
    return all_files

def filter_csv(file_path):
    «»»Фільтрує дані у файлі та перезаписує його.»»»
    If not os.path.exists(file_path):
        print(f» Файл {file_path} не знайдено, пропускаємо.»)
        return

    try:
        df = pd.read_csv(file_path)

        if df.empty:
            print(f» Файл {file_path} порожній, пропускаємо.»)
            return

        def slice_data_by_fields(group):
            filtered_rows = []
            one_way_group = group[group[«is_one_way»] == «Yes»]
            two_way_group = group[group[«is_one_way»] == «No»]

            for (origin, destination), subgroup in one_way_group.groupby([«origin», «destination»]):
                desktop_group = subgroup[subgroup[«platform»] == «desktop»]
```

```
                app_group = subgroup[subgroup[«platform»] == «app»]
                if len(desktop_group) >= 3:
                    filtered_rows.append(desktop_group.head(3))
                if len(app_group) >= 3:
                    filtered_rows.append(app_group.head(3))

            for (origin, destination), subgroup in
two_way_group.groupby([«origin», «destination»]):
                desktop_group = subgroup[subgroup[«platform»] ==
«desktop»]
                app_group = subgroup[subgroup[«platform»] == «app»]
                if len(desktop_group) >= 3:
                    filtered_rows.append(desktop_group.head(3))
                if len(app_group) >= 3:
                    filtered_rows.append(app_group.head(3))

            return pd.concat(filtered_rows, ignore_index=True) if
filtered_rows else pd.DataFrame()

        filtered_data = df.groupby(«pos»,
group_keys=False).apply(slice_data_by_fields).reset_index(drop=True)

        # Перезаписуємо той самий файл
        filtered_data.to_csv(file_path, index=False)
        print(f» Файл перезаписано: {file_path}»)

    except Exception as e:
        print(f» Помилка обробки {file_path}: {e}»)

def process_today_files():
    «»»Обробляє всі файли у динамічній папці за сьогодні.»»»
    Today_folder = get_today_folder()  # Отримуємо шлях до папки
    today_files = get_today_files(today_folder)

    if not today_files:
        print(« Немає файлів у сьогоднішній папці.»)
        return

    for file in today_files:
        filter_csv(file)

# Викликаємо функцію для обробки
process_today_files()
```

# ДОДАТОК Е

Лістинг коду sftp_to_postgres.py

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
import sys
import os

# Додаємо шлях до модулів, якщо вони не знаходяться у стандартній папці
sys.path.append('/Users/victoriashalahan/airflow/dags/modules')

# Імпортуємо функції з модулів
from modules.sftp import execute
from modules.sort import filter_csv
from modules.output_format import format_output_function
from modules.process_data import process_and_load_data

# 🔷 **Налаштування DAG**
default_args = {
    «owner»: «airflow»,
    «depends_on_past»: False,
    «start_date»: datetime(2024, 1, 1),
    «retries»: 1,
    «retry_delay»: timedelta(minutes=5),
}

# 🔷 **Функція для обробки файлів з Xcom**
def process_files_from_xcom(task_instance):
    «»»Отримує список файлів із Xcom і обробляє кожен із них.»»»
    File_paths  =  task_instance.xcom_pull(task_ids=»download_files»,
key=»downloaded_files»)

    if not file_paths:
        print(«⚠️ Немає файлів у Xcom для обробки.»)
        return

    for file_path in file_paths:
        filter_csv(file_path)  # Викликаємо функцію з sort.py

# 🔷 **Створення DAG**
with DAG(
    dag_id=»sftp_to_postgres»,
    default_args=default_args,
    schedule=»@daily»,  # Запуск щодня о 00:00
    catchup=False,
) as dag:

    # 🛠️ 1️⃣ **Завантаження файлів з SFTP**
    download_task = PythonOperator(
```

```
        task_id=»download_files»,
        python_callable=execute,
        op_args=[«{{ ds }}»]  # Поточна дата для завантаження файлів
    )

    # 2️⃣ **Сортування файлів**
    sort_task = PythonOperator(
        task_id=»sort_file»,
        python_callable=process_files_from_xcom,
        provide_context=True
    )

    # 3️⃣ **Форматування виводу**
    output_task = PythonOperator(
        task_id=»output_file»,
        python_callable=format_output_function  # Викликаємо функцію
без аргументів
    )

    # 4️⃣ **Обробка файлів і завантаження у PostgreSQL**
    process_task = PythonOperator(
        task_id=»process_and_load_files»,
        python_callable=process_and_load_data  # Функція, яка шукає
файли та завантажує їх у БД
    )

    # 🔄 **Визначення порядку виконання завдань**
    download_task >> sort_task >> output_task >> process_task
```

# ДОДАТОК Ж

Лістинг коду process_data.py

```
import pandas as pd
import os
import logging
from datetime import datetime
from sqlalchemy import create_engine

# Налаштування логування
log = logging.getLogger(__name__)

def process_and_load_data():
    downloads_folder = os.path.expanduser("~/Downloads")
    today_folder = datetime.today().strftime('%Y-%m-%d')
    target_folder = os.path.join(downloads_folder, today_folder)

    if not os.path.exists(target_folder):
        log.error(f"❌ Папка {target_folder} не знайдена.")
        return

    log.info(f"📂 Шукаємо CSV-файли в: {target_folder}")

    db_name = "ag_air_flights"
    username = "victoriashalahan"
    password = "shalahan"  # ⚠️ Використовуй змінні середовища для безпеки
    host = "localhost"
    port = "5432"

    # Створення підключення до PostgreSQL через SQLAlchemy
    engine = create_engine(f'postgresql+psycopg2://{username}:{password}@{host}:{port}/{db_name}')

    csv_files = [f for f in os.listdir(target_folder) if f.endswith('.csv')]

    if not csv_files:
        log.warning("⚠️ CSV-файли не знайдені.")
        return

    for file_name in csv_files:
        file_path = os.path.join(target_folder, file_name)

        try:
            df = pd.read_csv(file_path)
            log.info(f"📊 {file_name}: {len(df)} рядків, {len(df.columns)} колонок.")
```

```
            with engine.connect() as conn:
                # Використання engine для з'єднання з PostgreSQL
                df.to_sql('flights_data',                    con=conn,
if_exists='append', index=False, method='multi')
                log.info(f"✅ Файл {file_name} успішно завантажено в
базу.")

        except Exception as e:
            log.error(f"❌  Помилка  обробки  {file_name}:  {e}",
exc_info=True)
```

# ДОДАТОК К

Лістинг коду output_format.py

```
import csv
import os
from datetime import datetime

def get_today_folder():
    """Формує шлях до папки за сьогоднішню дату в 'Downloads'."""
    home_dir = os.path.expanduser("~")
    downloads_dir = os.path.join(home_dir, "Downloads")
    today = datetime.today().strftime('%Y-%m-%d')
    folder_path = os.path.join(downloads_dir, today)
    return folder_path

def validate_csv(input_path):
    """Перевіряє дані у CSV-файлі та перезаписує його."""
    if not os.path.exists(input_path):
        print(f"Error: File '{input_path}' not found.")
        return

    required_fields = {
        "sourceOTA",   "url",   "check_in_date",   "check_out_date",
"origin", "destination", "rout_type", "pos",
        "outbound_flight_no",                    "inbound_flight_no",
"outbound_departure_date", "outbound_departure_time",
        "outbound_arrival_date",             "outbound_arrival_time",
"inbound_departure_date", "inbound_departure_time",
        "inbound_arrival_date",               "inbound_arrival_time",
"Outbound_Carrier", "Inbound_Carrier", "id",
        "observation_date",      "observation_time",      "currency",
"price_outbound", "price_inbound",
        "price_outbound_USD",                    "price_inbound_USD",
"outbound_flight_duration", "Inbound_flight_duration",
        "SRP_Rank", "screenshot", "is_one_way"
    }

    one_way_empty_fields = {
        "check_out_date",                        "inbound_flight_no",
"inbound_departure_date", "inbound_departure_time",
        "inbound_arrival_date",               "inbound_arrival_time",
"Inbound_Carrier", "price_inbound",
        "price_inbound_USD", "Inbound_flight_duration"
    }

    def validate_decimal(value):
        """Перевіряє, чи є значення десятковим числом."""
        try:
            float(value)
            return True
        except ValueError:
            return False
```

```
    with open(input_path, mode="r", encoding="utf-8") as file:
        reader = csv.DictReader(file)
        normalized_headers = {field.lower(): field for field in
reader.fieldnames}

        fieldnames = ["zero_one", "comment"] + [field for field in
reader.fieldnames if field not in ["zero_one", "comment"]]
        validated_rows = []

        for row in reader:
            row["zero_one"] = "1"
            row["comment"] = ""
            is_one_way = row.get("is_one_way") == "Yes"
            platform = row.get("platform", "").strip().lower()

            for field in required_fields:
                normalized_field                                    =
normalized_headers.get(field.lower(), field)
                value = row.get(normalized_field, "").strip()

                if not value:
                    if    not    (is_one_way    and    field    in
one_way_empty_fields):
                        # Пропускаємо помилку, якщо поле 'url' і
platform = 'app'
                        if field == "url" and platform == "app":
                            continue
                        row["zero_one"] = "0"
                        row["comment"] += f"{field} was not extracted;
"
                elif field in {"price_outbound", "price_inbound",
"price_outbound_USD", "price_inbound_USD"}:
                    if not validate_decimal(value):
                        row["zero_one"] = "0"
                        row["comment"] += f"{field} was extracted
incorrectly; "

            validated_rows.append(row)

    with open(input_path, mode="w", encoding="utf-8", newline="") as
file:
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(validated_rows)

    print(f"Validation complete. File updated: {input_path}")

def format_output_function():
    """Шукає CSV-файли у папці за поточною датою та валідує їх
(перезаписує)."""
    input_folder = get_today_folder()
```

```
if not os.path.exists(input_folder):
    print(f"Error: Folder '{input_folder}' not found.")
    return

files_to_process = []

for filename in os.listdir(input_folder):
    if filename.endswith(".csv"):
        input_path = os.path.join(input_folder, filename)
        files_to_process.append(input_path)

if not files_to_process:
    print(f"No CSV files found in {input_folder}.")
    return

for input_path in files_to_process:
    print(f"Processing file: {input_path}")
    validate_csv(input_path)
```

# ДОДАТОК Л

Лістинг коду graph.py

```
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter

# Зчитування CSV-файлу
df = pd.read_csv('/Users/victoriashalahan/start')

# Заміна 'all good' для рядків без помилок
df['comment'] = df.apply(lambda row: 'all good' if row['0/1'] == 1
else row['comment'], axis=1)

# Збір усіх помилок
all_comments = []

for comment in df['comment']:
    if comment == 'all good':
        all_comments.append('all good')
    else:
        split_errors = [e.strip() for e in comment.split(';') if
e.strip()]
        all_comments.extend(split_errors)

# Підрахунок кількості кожної унікальної помилки
counts = Counter(all_comments)
total_records = len(df)
percentages = {error: (count / total_records) * 100 for error, count
in counts.items()}

# Підготовка даних до графіка
labels = list(percentages.keys())
values = list(percentages.values())
count_values = [counts[label] for label in labels]

# Кольори: зелений для 'all good', інші — з палітри
colors = ['green' if label == 'all good' else plt.cm.tab20(i) for i,
label in enumerate(labels)]

# Отримання провайдера та платформ
provider = df['sourceOTA'].iloc[0] if 'sourceOTA' in df.columns else
'Unknown provider'
platform_list = df['platform'].unique() if 'platform' in df.columns
else ['Unknown platform']

# Формування підпису: платформа / платформи
if len(platform_list) == 1:
    platform_text = f"Платформа: {platform_list[0]}"
else:
    platforms = ', '.join(platform_list)
```

```
    platform_text = f"Платформи: {platforms}"

# Побудова заголовка
title = f"Аналіз знайдених помилок\nПровайдер: {provider}, {platform_text}"

# Побудова кругової діаграми
fig, ax = plt.subplots(figsize=(8, 8))
wedges, texts, autotexts = ax.pie(
    values,
    labels=None,
    autopct='%1.1f%%',
    colors=colors,
    startangle=140,
    textprops=dict(color="white")
)

# Легенда
ax.legend(
    wedges,
    [f'{label} ({count_values[i]} записів)' for i, label in enumerate(labels)],
    title="Тип помилки / статус",
    loc="center left",
    bbox_to_anchor=(1, 0, 0.5, 1),
    frameon=False
)

plt.setp(autotexts, size=10, weight="bold")
ax.set_title(title)
plt.tight_layout()
plt.show()
```