ЛАБОРАТОРНА РОБОТА 2

Тема: Підготовка та очищення бізнес даних

Підготовка та очищення даних (Data Preparation & Cleaning) — це процес перетворення «сирих» даних у чистий, структурований формат, придатний для аналізу. Основною метою його є забезпечення достовірності даним і має вплив у залежності конкретних бізнес-цілей.

1. **Підвищення точності прогнозів**

Бізнес будує стратегії на основі цифр. Якщо в базі даних клієнтів один і той самий покупець записаний як «Khoma Brut» та «Хома Брут», алгоритм сприйме це як двох різних людей. Очищення допомагає уникнути такого дублювання, що дає реальну картину продажів.

2. **Економія часу та ресурсів**

Аналітики витрачають до 80% свого часу саме на підготовку даних і лише 20% на сам аналіз. Автоматизація очищення дозволяє швидше переходити до прийняття рішень.

3. **Виявлення прихованих проблем**

Під час підготовки даних часто спливають системні помилки в бізнес-процесах. Наприклад, у стовпчику «Ціна» виставлено нуль. Найчастішою причиною є технічний збій, помилка інтеграції: дані передаються з однієї системи в іншу і на певному її етапі сталась помилка - платіжний шлюз не передав суму у систему обліку. Як результат замовлення є, товар зі складу списано, а в графі «Ціна» стоїть системний нуль за замовчуванням. Ще однією причиною є людський фактор: помилка неминуче виникає під час внесення даних вручну. Наприклад, менеджер на час створення картки нового товару ще не знав його ціну, але потім забув її внести. Зрозуміло, що поодинокий екземпляр не може йти у продаж за ціною 0. Дещо складнішим випадком є знайомі нам маркетингові акції та подарунки: нулі в ціні не є помилкою під час акції типу «1+1=3», тобто третій однотипний товар у чеку проходить за ціною 0, або клієнт використав бонусні бали, які повністю покрили вартість покупки певного товару. У цьому випадку бізнес-аналіз даних щодо попиту на певний товар має відокремити такі «нульові» транзакції від звичайних продажів — реальний попит на товар не збільшується за цей рахунок.

Процес перетворення сирих, інколи кажуть брудних даних на дані, які можуть бути використані в бізнес-аналітиці зветься Data Wrangling або Data Preprocessing і характеризується певною стратегією послідовності операцій. Такими операціями є:

1. **Збір та консолідація (Data Integration)**

В інформаційній системі навіть одного підприємства дані можуть знаходитись на різних носіях: різні СУБД та SQL СУБД, електроні таблиці, файлові репозитарії з різними форматами (txt, csv, json), хмарні сховища, тощо. Тому, перед виконанням будь-яких операцій бізнес-аналітики необхідно об’єднати усі необхідні дані з усіх доступних джерел.

2. **Очищення даних (Data Cleaning)**

Необхідно очистити дані від сміття: робота з нулями, пропусками, дублікатами, тощо. Як правило, основними операціями на цьому етапі є пошук та видалення повторів, заповнення пустих клітинок, наприклад, середнім значенням або значеннями за замовчуванням та виправлення помилок пов’язаних з уніфікацією назв та імен, наприклад, «Київ» і «КИЇВ» мають ідентифікуватись якимось одним варіантом. У результаті дані стають однорідними та достовірними.

3. **Форматування та приведення типів (Type Casting)**

Дані можуть надходити та зберігатись на носіях за різними форматами та типам даних. Так, у кожній країні світи свій стандарт запису дати, тому як в офіційних документах, так і в інформаційних системах зустрічаються формати дати: "14/02/2026", "Feb 14, 2026", "2026.02.14 16:50". Фінансові дані та показники в різних документах та потокових даних можуть супроводжуватись зайвими символами, що призначено для правильного та однозначного сприйняття людиною, але унеможливлюють безпосереднє використання в бізнес-аналізі: "$1,250.50", "1 500,00 грн", "12.5%". У базах даних певні параметри, що мають тільки два значення, можуть позначати свої статуси у тестовому форматі замість логічного (boolean): "стать"

позначається як "Чоловік/Жінка", "підписка" має значення "Так/Ні", тощо. Частим випадком є втрата «старших» нулів при певних видах кодування, наприклад, на поштовому індексі, коді оператору телефонії, в коді товару - мало бути «00541», але при збережені у форматі int отримуємо «541» і втрату інформації про те, скільки нулів було попереду. Застосування сучасних форматів даних та API, що їх використовують може приводити до випадків, коли дані надсилаються у вигляді єдиної структури або єдиного пакету, який сприймається і оброблюється як ціле. Наприклад, застосування формату JSON для адреси: {"city": "Kyiv", "zip": "01001", "street": "Khreshchatyk"}. Це ускладнює, а інколи унеможливлює безпосереднє їх використання, наприклад, фільтрацію за містом не може бути виконано безпосередньо, тому що необхідні дані є складовою певної структури, необхідні дані спочатку потрібно «витягнути». Таким чином, перед виконанням бізнес-аналізу дані мають бути приведені до єдиного формату, єдиного типу даних з урахуванням усіх нюансів і їх призначення.

4. **Нормалізація та масштабування (Scaling)**

Приведення до єдиної шкали або масштабування необхідне тоді, коли необхідно виконати порівняння даних різної природи, наприклад, одні вимірюються в мільйонах, а інші у відсотках. Припустимо, банк оцінює надійність клієнта, тобто квиконує кредитний скоринг. В якості першого критерію використовується річний дохід (діапазон від 0 до 1 млн.), другий критерій - кількість прострочених платежів (від 0 до 5). Якщо просто додавати ці значення, то «дохід» буде повністю поглинати «прострочення». Такий підхід приводить до тог, що клієнт з доходом у 1 млн. і 5 простроченнями виглядає привабливіше для банку, ніж клієнт із доходом 900 тис. і 0 прострочок, хоча з точки зору ризику усе навпаки. Якщо з точки зору банку ці критерії мають однакову вагу, то їх значення слід нормалізувати — перетворити в бали від 0 до 100 і, таким чином, покращити відносний вплив на результат аналізу.

5. **Конструювання ознак (Feature Engineering)**

Це найбільш творча частина роботи аналітика. Аналітик створює нові дані на основі наявних задля виявлення глибших сенсів. Найпоширенішим є використання часових ознак (Time-based Features). Так, будь-яка дата (наприклад, 2024-12-25) для алгоритму є певним числовим значенням. Аналітик може розкласти її на змістовні частини. До дати транзакції ще визначити та додати день тижня (понеділок, субота чи неділя). Наприклад, для ресторанів будні характеризуються придбанням бізнес-ланчів, а у вихідні вищим попитом на алкоголь. Також важливим є визначення таких добових категорій, як: ранок, обід, вечір, ніч. Насамперед це може бути пов’язано з використанням певного типу та об’ємів електроживлення. Також на адекватність висновків впливає визначення чи є день святковим, а інколи і яке саме свято. Зрозумілий прогноз підвищеного попиту на Різдво, Пасху та 14 лютого, але асортимент попиту буде відрізнятись і на окремі групи товарів суттєво. Одним із найбільш яскравих прикладів Feature Engineering є RFM-аналіз, який дозволяє сегментувати клієнтів на основі їхньої минулої купівельної поведінки. Recency (Давність) вказує на давність останньої покупки клієнтом і тим самим на ймовірність наступної покупки. Frequency (Частота) вказує як часто клієнт купує і це рівень лояльності клієнта. Monetary (Гроші) вказує на загальну суму витрат клієнта або на його цінність для бізнесу. Кожному клієнту присвоюється бал (зазвичай від 1 до 5) за кожним із трьох показників: 5 — найкращий показник, 1 — найгірший показник. Таким чином кожен покупець отримує «код», наприклад 555 (ідеальний клієнт) або 111 (втрачений клієнт). Основними сегментами клієнтів є: 555, 554 - найкращі клієнти, їм не треба заважати але надавати ранній доступ до нових продуктів; 455, 355 - стабільні покупці, треба не втрачати з ними контакт, просити відгуки і пропонувати програму лояльності; 222, 211 — клієнти, що «засинають», купували раніше, але давно не поверталися, необхідно розробити персональну знижку; 155, 144 - колись витрачали багато, але зараз не купують, потрібно терміново зв'язатися, надати дуже вигідні пропозиції; 511, 411 — новачки, зробили першу покупку, потрібно пояснити переваги бренду. Однією із особливостей RFM — для отримання важливих характеристик йому не потрібен складний AI, а як результат — клієнтам не розсилаються однакові пропозиції, кращим клієнтам надають ранній доступ до новинок, а клієнту, що йде, надають знижки. Але RFM не підходить для бізнесів з одноразовими покупками, наприклад, квартири, весільні сукні, тому як

показник Frequency (Частота) не має чіткого сенсу, а для ресторанів, інтернет-магазинів косметики RFM практично ідеальний інструмент.

6. Перевірка на аномалії та валідація (Data Validation)

Перед тим, як дані потраплять у звіт до керівництва або в модель машинного навчання вони мають пройти перевірку на аномалії та валідацію. Валідація — це перевірка даних на здорову логіку та фізичну можливість. Наприклад, дата доставки не може бути раніше дати замовлення або вік клієнта не може бути "150 років" або "-5". Також, сума транзакції в магазині не може бути від'ємною, за виключенням щодо повернення, яке має бути позначене спеціальним типом. Важливим є перевірка на унікальність, наприклад, податкового номеру або номеру паспорта, які не можуть дублюватись у двох різних клієнтів. До цього етапу відносяться і перевірки на статистичні аномалії та статистичну консистентність, наприклад, на наявність неймовірно великих значень отриманих надходжень, суттєве відхилення від середнього, різке зростання попиту за показником замовлень в Інтернет магазині. Цей етап має гарантувати, що звіт з аналітики не викличе паніки або хибної ейфорії у стейкхолдерів.

Практичний приклад 1

На прикладі біржових даних з Yahoo Finance виконується пошук певних аномалій даних:

1. Пропуски у часовому ряді
2. Дні з нульовим об’ємом
3. Дні без змін ціни
4. Аномальні зміни ціни
5. Аномально великі пропуски торгів

Для кращого сприйняття результати пошуку візуалізуються у вигляді діаграм.

```
import yfinance as yf
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Тікери компаній
tickers = [
    'AAPL',        # Apple
    'GOOGL',       # Alphabet Inc.
    'MSFT',        # Microsoft
    'TSLA',        # Tesla
    'AMZN',        # Amazon
    'NVDA',        # NVIDIA
    'META'         # Meta
]

# Завантаження даних за 5 років
end_date = datetime.now()
start_date = end_date - timedelta(days=5*365)

data = {}
print("=== ЗАВАНТАЖЕННЯ ДАНИХ ===\n")
for ticker in tickers:
    try:
        df = yf.download(ticker, start=start_date, end=end_date, progress=False)
        if not df.empty:
            data[ticker] = df
            print(f" {ticker}: {len(df)} днів даних")
        else:
            print(f" {ticker}: Дані відсутні")
    except Exception as e:
        print(f" {ticker}: Помилка - {e}")

# 1. ПОШУК ПРОПУСКІВ У ЧАСОВОМУ РЯДІ
print("\n=== 1. АНАЛІЗ ПРОПУСКІВ У ЧАСОВОМУ РЯДІ ===\n")

for ticker, df in data.items():
    all_days = pd.date_range(start=df.index.min(), end=df.index.max(), freq='D')
    trading_days = len(df)
    calendar_days = len(all_days)

    missing_days = calendar_days - trading_days
    missing_pct = (missing_days / calendar_days) * 100

    print(f"{ticker}:")
    print(f"  Торгових днів: {trading_days}")
```

```
    print(f"  Календарних днів: {calendar_days}")
    print(f"  Пропущено днів: {missing_days} ({missing_pct:.2f}%)")

    if missing_days > 0:
        existing_dates = set(df.index.date)
        all_dates = set(all_days.date)
        missing_dates = sorted(list(all_dates - existing_dates))[:10]
        print(f"  Приклади пропущених днів: {[str(d) for d in missing_dates]}")

# 2. АНАЛІЗ ДНІВ З НУЛЬОВИМ ОБ'ЄМОМ
print("\n=== 2. АНАЛІЗ ДНІВ З НУЛЬОВИМ ОБ'ЄМОМ ===\n")

for ticker, df in data.items():
    if 'Volume' in df.columns:
        if isinstance(df['Volume'], pd.DataFrame):
            volume_series = df['Volume'].iloc[:, 0]
        else:
            volume_series = df['Volume']

        zero_volume = int((volume_series == 0).sum())
        zero_volume_pct = (zero_volume / len(df)) * 100

        print(f"{ticker}: {zero_volume} днів з нульовим об'ємом ({zero_volume_pct:.2f}%)")

        if zero_volume > 0:
            zero_days = df.index[volume_series == 0]
            print(f"  Дати: {[d.strftime('%Y-%m-%d') for d in zero_days[:5]]}")

# 3. АНАЛІЗ ДНІВ БЕЗ ЗМІНИ ЦІНИ
print("\n=== 3. АНАЛІЗ ДНІВ БЕЗ ЗМІНИ ЦІНИ ===\n")

for ticker, df in data.items():
    if 'Close' in df.columns and 'Open' in df.columns:
        if isinstance(df['Close'], pd.DataFrame):
            close_series = df['Close'].iloc[:, 0]
            open_series = df['Open'].iloc[:, 0]
        else:
            close_series = df['Close']
            open_series = df['Open']

        price_unchanged = int((close_series == open_series).sum())
        price_unchanged_pct = (price_unchanged / len(df)) * 100

        print(f"{ticker}: {price_unchanged} днів без зміни ціни ({price_unchanged_pct:.2f}%)")

        if price_unchanged > 0:
            flat_days = df.index[close_series == open_series]
            print(f"  Приклади: {[d.strftime('%Y-%m-%d') for d in flat_days[:5]]}")

# 4. АНАЛІЗ АНОМАЛЬНИХ ЗМІН ЦІНИ
print("\n=== 4. АНАЛІЗ АНОМАЛЬНИХ ЗМІН ЦІНИ ===\n")

for ticker, df in data.items():
    if 'Close' in df.columns:
        if isinstance(df['Close'], pd.DataFrame):
            close_series = df['Close'].iloc[:, 0]
        else:
            close_series = df['Close']

        returns = close_series.pct_change() * 100
        extreme_mask = (abs(returns) > 20).fillna(False)

        extreme_returns = int(extreme_mask.sum())
        extreme_returns_pct = (extreme_returns / len(df)) * 100

        print(f"{ticker}: {extreme_returns} днів зі зміною >20% ({extreme_returns_pct:.2f}%)")

        if extreme_returns > 0:
            extreme_dates = returns.index[extreme_mask]
            print(f"  Дати: {[d.strftime('%Y-%m-%d') for d in extreme_dates[:5]]}")

            for date in extreme_dates[:3]:
                ret_value = returns.loc[date]
                print(f"    {date.strftime('%Y-%m-%d')}: {ret_value:.1f}%")

# 5. ВІЗУАЛІЗАЦІЯ "ПРИХОВАНИХ" ПРОПУСКІВ
print("\n=== 5. ВІЗУАЛІЗАЦІЯ ПРИХОВАНИХ ПРОПУСКІВ ===\n")

fig, axes = plt.subplots(len(data), 1, figsize=(15, 4*len(data)))
if len(data) == 1:
    axes = [axes]

for idx, (ticker, df) in enumerate(data.items()):
    ax = axes[idx]

    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    ax.plot(df.index, close_series, 'b-', linewidth=1, alpha=0.7, label='Ціна закриття')
```

```
    if 'Volume' in df.columns:
        if isinstance(df['Volume'], pd.DataFrame):
            volume_series = df['Volume'].iloc[:, 0]
        else:
            volume_series = df['Volume']

        zero_vol_days = df.index[volume_series == 0]
        if len(zero_vol_days) > 0:
            ref_value = close_series.median()
            ax.scatter(zero_vol_days, [ref_value] * len(zero_vol_days),
                       color='red', s=50, marker='o', alpha=0.7,
                       label=f'Нульовий об\'єм ({len(zero_vol_days)})')

    if 'Open' in df.columns:
        if isinstance(df['Open'], pd.DataFrame):
            open_series = df['Open'].iloc[:, 0]
        else:
            open_series = df['Open']

        flat_days = df.index[close_series == open_series]
        if len(flat_days) > 0:
            ref_value = close_series.median() * 0.95
            ax.scatter(flat_days, [ref_value] * len(flat_days),
                       color='orange', s=30, marker='s', alpha=0.7,
                       label=f'Ціна не змінилась ({len(flat_days)})')

    returns = close_series.pct_change() * 100
    extreme_mask = (abs(returns) > 20).fillna(False)
    extreme_days = returns.index[extreme_mask]
    if len(extreme_days) > 0:
        ref_value = close_series.median() * 1.05
        ax.scatter(extreme_days, [ref_value] * len(extreme_days),
                   color='purple', s=80, marker='^', alpha=0.7,
                   label=f'Аномальна зміна ({len(extreme_days)})')

    ax.set_title(f'{ticker} - приховані проблеми з даними', fontsize=12, fontweight='bold')
    ax.set_ylabel('Ціна ($)')
    ax.legend(loc='upper left')
    ax.grid(True, alpha=0.3)

    stats_text = f"Днів: {len(df)}\n"
    stats_text += f"Нульовий об'єм: {len(zero_vol_days) if 'Volume' in df.columns else 0}\n"
    stats_text += f"Ціна не змін.: {len(flat_days) if 'Open' in df.columns else 0}\n"
    stats_text += f"Аномалії: {len(extreme_days)}"

    ax.text(0.02, 0.98, stats_text, transform=ax.transAxes, fontsize=9,
            verticalalignment='top', bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.tight_layout()
plt.show()

# 6. АНАЛІЗ РЕГУЛЯРНОСТІ ТОРГІВ
print("\n=== 6. АНАЛІЗ РЕГУЛЯРНОСТІ ТОРГІВ ===\n")

for ticker, df in data.items():
    # Розрахунок інтервалів
    date_diff = df.index.to_series().diff().dt.days
    intervals = date_diff.dropna()

    print(f"{ticker}:")
    print(f"  Мінімальний інтервал: {intervals.min():.0f} днів")
    print(f"  Максимальний інтервал: {intervals.max():.0f} днів")
    print(f"  Середній інтервал: {intervals.mean():.1f} днів")

    # Шукаємо аномально великі інтервали (> 5 днів)
    large_gaps = intervals[intervals > 5]
    if len(large_gaps) > 0:
        print(f"    Знайдено {len(large_gaps)} великих пропусків (>5 днів):")

        for i, (idx, gap_size) in enumerate(large_gaps.items()):
            if i >= 3:  # Показуємо тільки перші 3
                break
            # idx - це індекс (дата) з large_gaps
            gap_date = idx
            print(f"    {gap_date.strftime('%Y-%m-%d')}: пропуск {gap_size:.0f} днів (до цієї дати)")

# 7. ПІДСУМКОВИЙ ЗВІТ
print("\n=== ПІДСУМКОВИЙ ЗВІТ ПРО ПРОПУСКИ ===\n")

summary = []
for ticker, df in data.items():
    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    if 'Volume' in df.columns:
        if isinstance(df['Volume'], pd.DataFrame):
            volume_series = df['Volume'].iloc[:, 0]
        else:
```

```
            volume_series = df['Volume']
        zero_volume = int((volume_series == 0).sum())
    else:
        zero_volume = 0

    if 'Open' in df.columns:
        if isinstance(df['Open'], pd.DataFrame):
            open_series = df['Open'].iloc[:, 0]
        else:
            open_series = df['Open']
        no_price_change = int((close_series == open_series).sum())
    else:
        no_price_change = 0

    returns = close_series.pct_change()
    extreme_returns = int((abs(returns) > 0.2).fillna(False).sum())

    # Розрахунок максимального інтервалу
    intervals = df.index.to_series().diff().dt.days.dropna()
    max_gap = int(intervals.max()) if not intervals.empty else 0

    row = {
        'Ticker': ticker,
        'Total_Days': len(df),
        'Calendar_Days': len(pd.date_range(start=df.index.min(), end=df.index.max(), freq='D')),
        'Missing_Days': int(len(pd.date_range(start=df.index.min(),end=df.index.max(),freq='D'))-len(df)),
        'Zero_Volume': zero_volume,
        'No_Price_Change': no_price_change,
        'Extreme_Returns': extreme_returns,
        'Max_Gap_Days': max_gap
    }
    summary.append(row)

summary_df = pd.DataFrame(summary)
print(summary_df.to_string(index=False))

print("  Аналіз завершено:")
print("  • Calendar_Days vs Total_Days — реальні пропуски в календарі")
print("  • Zero_Volume — дні без торгів (фактичні пропуски)")
print("  • Max_Gap_Days — найбільший розрив між торговими днями")
```

Практичний приклад 2

Розрахунок, аналіз та нормалізація базових фінансових метрик

Денна звичайна (арифметична) прибутковість визначається наступним чином:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

де: P_t — ціна в день t

P_{t-1} — ціна попереднього дня

Отримані результати часто представляють у відсотках, наприклад, 0.02 = +2%. Основною перевагою цієї метрики є її інтуїтивне розуміння та прозорість в інтерпретації, а також вона є зручною для короткострокових прогнозів. Метрика не адитивна в часі. Тобто для визначення значення метрики за певний час не можна застосовувати звичайне додавання значень цієї метрики за окремі його проміжки.

Денна логарифмічна прибутковість визначається за формулою:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

Ця метрика є адитивною величиною, тому загальна величина може бути розрахована сума значень за менші проміжки:

$$r_{total} = r_1 + r_2 + \ldots + r_n$$

Ковзні середні спрощують і згладжують коливання цін, зменшуючи шум і дають змогу визначити поточній тренд ринку. Зменшення шуму (стрибків графіка ціни в різні боки) на графіку дає більш чітку картину того, що відбувається. Незважаючи на те, що ковзні середні корисні для отримання інформації, вони не передбачають майбутні тренди, а підтверджують вже існуючі.

Ковзна середня позначається як MA - скорочення від англ. Moving Average – “ковзна середня”. Таймфрейми в декілька днів або тижнів вважаються “високими”, а таймфрейм тривалістю 5-15 хвилин – “низьким таймфреймом”. 20-денне ковзне середнє (MA20) визначається як

$$MA_{20}(t) = \frac{1}{20}\sum_{i=0}^{19} P_{t-i}$$

Міра того, як змінюється ціна активу з часом, зветься волатильністю. Волатильність є ключовим поняттям в інвестуванні та фінансах. Якщо кажуть, що актив має високу волатильність, це означає, що його ціна часто і суттєво змінюється — як вгору, так і вниз. Наприклад, волатильність інвестиційного фонду — важливий фактор при його виборі, адже вона вказує на потенційні прибутки (або збитки), які може отримати інвестор. Часто це поняття обговорюється в контексті консервативних чи агресивних інвестиційних стратегій: низька волатильність означає стабільність, а висока — можливість вищого прибутку, але з більшим ризиком.

Математично волантильність визначається як стандартне відхилення доходності:

$$\sigma_{20}(t) = \sqrt{\frac{1}{19}\sum_{i=0}^{19}(r_{t-i} - \bar{r})^2}$$

де *r* — логорифм доходності (прибутковості).

RSI (Relative Strength Index, індекс відносної сили) з періодом 14 (днів/свічок) — це найпопулярніший технічний осцилятор, що вимірює швидкість та зміну цін за останні 14 періодів, наприклад, днів на денному графіку. Він коливається від 0 до 100. Класичний період 14 днів вважається стандартом, що балансує між чутливістю та надійністю сигналів. Якщо його значення >70, то він характеризується як перекупленість: актив купували занадто активно, у подальшому можливе зниження ціни або корекція. Якщо значення <30, то характеризується як перепроданість: актив продавали занадто активно і у подальшому можливе зростання ціни (відскок). Рівень 50 - це центральна лінія, яка допомагає визначити загальний тренд. Основними сигналами є:

- дивергенція, ціна росте, а RSI знижується (або навпаки), вказує на можливу зміну тренду;
- прорив рівнів, перетин лінії 50 або вихід із зон 30/70.

Хоча 14 є стандартним, короткострокові трейдери можуть зменшувати період (наприклад, до 9 або 7) для швидшої реакції, а довгострокові — збільшувати (наприклад, 25) для більш гладкої кривої.

RSI найкраще працює в трендових ринках, але може давати хибні сигнали під час сильних трендів, тому його рекомендують використовувати разом з такими індикаторами, як об'єми або ковзне середнє.

RSI розраховується за наступним алгоритмом.

1. Обчислюються денні зміни ціни:

$$\Delta P_t = P_t - P_{t-1}$$

2. Визначаються *AvgGain* - середній gain (позитивні зміни) та *AvgLoss* середній loss (негативні зміни).

3. Обчислюється

$$RS = \frac{AvgGain}{AvgLoss}$$

4. Тепер визначається індекс:

$$RSI = 100 - \frac{100}{1 + RS}$$

```
# 8. НОРМАЛІЗАЦІЯ ТА ТРАНСФОРМАЦІЯ ФІНАНСОВИХ ПОКАЗНИКІВ
print("\n=== 8. НОРМАЛІЗАЦІЯ ТА ТРАНСФОРМАЦІЯ ФІНАНСОВИХ ПОКАЗНИКІВ ===\n")

from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
import seaborn as sns
from scipy import stats

# Вибираємо тікери з наявними даними
available_tickers = [t for t in tickers if t in data]

print(f"Доступні тікери: {available_tickers}")

# 8.1 РОЗРАХУНОК ФІНАНСОВИХ МЕТРИК
print("\n--- 8.1 Розрахунок фінансових метрик ---\n")

def calculate_financial_metrics(close_series, volume_series=None):
    """
    Розрахунок фінансових метрик на основі ціни закриття
    """
    df_metrics = pd.DataFrame(index=close_series.index)
    df_metrics['Close'] = close_series

    # Денна прибутковість (звичайна)
    df_metrics['Daily_Return'] = close_series.pct_change()

    # Логарифмічна прибутковість (краща для статистичного аналізу)
    df_metrics['Log_Return'] = np.log(close_series / close_series.shift(1))

    # Ковзне середнє (20 днів)
    df_metrics['MA_20'] = close_series.rolling(window=20, min_periods=1).mean()

    # Відхилення від ковзного середнього
    df_metrics['MA_20_Deviation'] = (close_series - df_metrics['MA_20']) / df_metrics['MA_20'] * 100

    # Волатильність (20-денна)
    df_metrics['Volatility_20'] = df_metrics['Daily_Return'].rolling(window=20).std() * np.sqrt(252)

    # RSI (Relative Strength Index)
    delta = close_series.diff()

    # Розділяємо прибутки та збитки
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    # Середні прибутки та збитки за 14 днів
    avg_gain = gain.rolling(window=14, min_periods=1).mean()
    avg_loss = loss.rolling(window=14, min_periods=1).mean()

    # Розраховуємо RS та RSI
    rs = avg_gain / avg_loss
    df_metrics['RSI'] = 100 - (100 / (1 + rs))

    # Об'єм торгів (якщо є)
    if volume_series is not None:
        df_metrics['Volume'] = volume_series
        df_metrics['Volume_MA'] = volume_series.rolling(window=20, min_periods=1).mean()
        df_metrics['Volume_Ratio'] = volume_series / df_metrics['Volume_MA']

    return df_metrics

# Розраховуємо метрики для кожного тікера
metrics_data = {}
for ticker in available_tickers:
    df = data[ticker]

    # Отримуємо Series для Close та Volume
    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    if 'Volume' in df.columns:
        if isinstance(df['Volume'], pd.DataFrame):
            volume_series = df['Volume'].iloc[:, 0]
```

```
        else:
            volume_series = df['Volume']
    else:
        volume_series = None

    metrics_data[ticker] = calculate_financial_metrics(close_series, volume_series)

    print(f"\n{ticker} - перші 5 рядків метрик:")
    print(metrics_data[ticker][['Close', 'Daily_Return', 'Log_Return', 'RSI']].head().round(4))

# 8.2 ВІЗУАЛІЗАЦІЯ РОЗПОДІЛІВ
print("\n--- 8.2 Візуалізація розподілів до нормалізації ---\n")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.ravel()

# Вибираємо один тікер для детальної демонстрації
demo_ticker = available_tickers[0]
df_metrics = metrics_data[demo_ticker]

# Гістограма денної прибутковості
axes[0].hist(df_metrics['Daily_Return'].dropna(), bins=50, alpha=0.7, color='blue', edgecolor='black')
axes[0].set_title(f'{demo_ticker} - Розподіл денної прибутковості')
axes[0].set_xlabel('Денна прибутковість')
axes[0].set_ylabel('Частота')
axes[0].axvline(x=0, color='red', linestyle='--', alpha=0.5)

# Q-Q plot для перевірки нормальності
stats.probplot(df_metrics['Daily_Return'].dropna(), dist="norm", plot=axes[1])
axes[1].set_title(f'{demo_ticker} - Q-Q plot (перевірка нормальності)')

# Box plot для виявлення викидів
df_metrics[['Daily_Return', 'Log_Return']].dropna().boxplot(ax=axes[2])
axes[2].set_title('Box plot - виявлення викидів')
axes[2].set_ylabel('Значення')

# Гістограма логарифмічної прибутковості
axes[3].hist(df_metrics['Log_Return'].dropna(), bins=50, alpha=0.7, color='green', edgecolor='black')
axes[3].set_title(f'{demo_ticker} - Розподіл логарифмічної прибутковості')
axes[3].set_xlabel('Логарифмічна прибутковість')
axes[3].set_ylabel('Частота')
axes[3].axvline(x=0, color='red', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# 8.3 НОРМАЛІЗАЦІЯ ДАНИХ
print("\n--- 8.3 Порівняння методів нормалізації ---\n")

def normalize_financial_data(df_metrics, method='minmax'):
    """
    Нормалізація фінансових показників різними методами
    """
    # Вибираємо колонки для нормалізації
    cols_to_normalize = ['Close', 'Volume', 'MA_20', 'Volatility_20'] if 'Volume' in df_metrics.columns
else ['Close', 'MA_20', 'Volatility_20']
    cols_to_normalize = [col for col in cols_to_normalize if col in df_metrics.columns]

    df_norm = df_metrics.copy()

    # Заповнюємо пропуски для нормалізації
    for col in cols_to_normalize:
        df_norm[col] = df_norm[col].fillna(method='ffill').fillna(method='bfill').fillna(0)

    if method == 'minmax':
        scaler = MinMaxScaler()
        df_norm[cols_to_normalize] = scaler.fit_transform(df_norm[cols_to_normalize])
        return df_norm, scaler, 'Min-Max (0-1)'

    elif method == 'standard':
        scaler = StandardScaler()
        df_norm[cols_to_normalize] = scaler.fit_transform(df_norm[cols_to_normalize])
        return df_norm, scaler, 'Standard (середнє=0, std=1)'

    elif method == 'robust':
        scaler = RobustScaler()
        df_norm[cols_to_normalize] = scaler.fit_transform(df_norm[cols_to_normalize])
        return df_norm, scaler, 'Robust (стійкий до викидів)'

    elif method == 'log':
        # Логарифмічна трансформація
        for col in cols_to_normalize:
            if (df_norm[col] > 0).all():
                df_norm[f'{col}_log'] = np.log1p(df_norm[col])
            else:
                # Якщо є від'ємні значення, використовуємо стандартизацію
                scaler = StandardScaler()
                df_norm[f'{col}_scaled'] = scaler.fit_transform(df_norm[[col]])
        return df_norm, None, 'Log трансформація'

# Демонструємо різні методи нормалізації
```

```
methods = ['minmax', 'standard', 'robust', 'log']
normalized_data = {}

for method in methods:
    df_norm, scaler, method_name = normalize_financial_data(df_metrics, method)
    normalized_data[method] = df_norm

    print(f"\nМетод: {method_name}")
    if 'Close' in df_norm.columns:
        close_col = 'Close' if 'Close' in df_norm.columns else 'Close_scaled' if 'Close_scaled' in
df_norm.columns else None
        if close_col:
            print(f"  Close - min: {df_norm[close_col].min():.4f}, max: {df_norm[close_col].max():.4f}")
    if 'Volume' in df_norm.columns:
        print(f"  Volume - min: {df_norm['Volume'].min():.4f}, max: {df_norm['Volume'].max():.4f}")

# 8.4 ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ НОРМАЛІЗАЦІЇ
print("\n--- 8.4 Візуалізація результатів нормалізації ---\n")

fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.ravel()

for idx, (method, df_norm) in enumerate(normalized_data.items()):
    ax = axes[idx]

    # Визначаємо, які колонки використовувати для візуалізації
    if 'Close' in df_norm.columns:
        close_col = 'Close'
    elif 'Close_scaled' in df_norm.columns:
        close_col = 'Close_scaled'
    elif 'Close_log' in df_norm.columns:
        close_col = 'Close_log'
    else:
        close_col = None

    if close_col:
        ax.plot(df_norm.index, df_norm[close_col], linewidth=1, label='Close')

    if 'Volume' in df_norm.columns:
        ax.plot(df_norm.index, df_norm['Volume'], linewidth=1, alpha=0.7, label='Volume')

    method_names = {'minmax': 'Min-Max', 'standard': 'Standard', 'robust': 'Robust', 'log': 'Log'}
    ax.set_title(f'{method_names[method]} нормалізація')
    ax.set_xlabel('Дата')
    ax.set_ylabel('Нормалізоване значення')
    ax.legend()
    ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# 8.5 ВПЛИВ НОРМАЛІЗАЦІЇ НА РІЗНІ ТІКЕРИ
print("\n--- 8.5 Порівняння нормалізованих рядів для різних тікерів ---\n")

# Нормалізуємо всі тікери одним методом для порівняння
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.ravel()

for idx, ticker in enumerate(available_tickers[:4]):  # Перші 4 тікери
    ax = axes[idx]

    # Отримуємо дані
    df = data[ticker]
    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    # Нормалізуємо (Min-Max)
    scaler = MinMaxScaler()
    close_norm = scaler.fit_transform(close_series.values.reshape(-1, 1)).flatten()

    ax.plot(close_series.index, close_norm, linewidth=1.5)
    ax.set_title(f'{ticker} - нормалізована ціна (Min-Max)')
    ax.set_xlabel('Дата')
    ax.set_ylabel('Нормалізована ціна')
    ax.grid(True, alpha=0.3)

    # Додаємо позначки проблемних місць
    if 'Volume' in df.columns:
        if isinstance(df['Volume'], pd.DataFrame):
            volume_series = df['Volume'].iloc[:, 0]
        else:
            volume_series = df['Volume']

        zero_vol_days = close_series.index[volume_series == 0]
        if len(zero_vol_days) > 0:
            for date in zero_vol_days[:3]:
                ax.axvline(x=date, color='red', alpha=0.3, linestyle='--')
                ax.text(date, 0.1, '⚠', fontsize=12, ha='center')
```

```
plt.tight_layout()
plt.show()

# 8.6 КОРЕЛЯЦІЙНИЙ АНАЛІЗ ПІСЛЯ НОРМАЛІЗАЦІЇ
print("\n--- 8.6 Кореляційний аналіз після нормалізації ---\n")

# Створюємо DataFrame з нормалізованими денними прибутковостями
returns_df = pd.DataFrame()

for ticker in available_tickers:
    df = data[ticker]
    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    # Денна прибутковість
    returns = close_series.pct_change()
    returns_df[ticker] = returns

# Видаляємо рядки з NaN
returns_df = returns_df.dropna()

# Кореляційна матриця
corr_matrix = returns_df.corr()

# Візуалізація
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=1, ax=ax, fmt='.2f')
ax.set_title('Кореляція денної прибутковості між тікерами')

plt.tight_layout()
plt.show()

print("Кореляційна матриця:")
print(corr_matrix.round(3))


# 8.7 ВИЯВЛЕННЯ АНОМАЛІЙ ЗА ДОПОМОГОЮ НОРМАЛІЗОВАНИХ ПОКАЗНИКІВ
print("\n--- 8.7 Виявлення аномалій за допомогою Z-оцінки ---\n")

for ticker in available_tickers:
    df = data[ticker]
    if isinstance(df['Close'], pd.DataFrame):
        close_series = df['Close'].iloc[:, 0]
    else:
        close_series = df['Close']

    # Розраховуємо Z-оцінку
    z_scores = np.abs((close_series - close_series.mean()) / close_series.std())

    # Аномалії (|z| > 3)
    anomalies = close_series[z_scores > 3]

    print(f"{ticker}: {len(anomalies)} аномалій за Z-оцінкою (>3σ)")

    if len(anomalies) > 0:
        anomaly_list = []
        for d, v in anomalies.head(3).items():
            date_str = d.strftime('%Y-%m-%d')
            anomaly_list.append(f"{date_str}: {v:.2f}")
        print(f"  Приклади: {anomaly_list}")

# 8.8 ПІДГОТОВКА ДАНИХ ДЛЯ МАШИННОГО НАВЧАННЯ
print("\n--- 8.8 Підготовка даних для машинного навчання ---\n")

def prepare_ml_dataset(df_metrics, target_col='Daily_Return', lookback=5):
    """
    Підготовка даних для прогнозування часових рядів
    """
    # Видаляємо рядки з NaN
    df_clean = df_metrics.dropna()

    # Створюємо лагові ознаки
    for i in range(1, lookback + 1):
        df_clean[f'Close_Lag_{i}'] = df_clean['Close'].shift(i)
        df_clean[f'Return_Lag_{i}'] = df_clean['Daily_Return'].shift(i)

    # Додаємо ковзні статистики
    df_clean['Close_MA_5'] = df_clean['Close'].rolling(5, min_periods=1).mean()
    df_clean['Close_MA_20'] = df_clean['Close'].rolling(20, min_periods=1).mean()
    df_clean['Volatility_5'] = df_clean['Daily_Return'].rolling(5, min_periods=1).std()

    # Цільова змінна (напрямок руху)
    df_clean['Target'] = (df_clean['Daily_Return'].shift(-1) > 0).astype(int)

    # Видаляємо рядки з NaN після створення ознак
    df_ml = df_clean.dropna()

    return df_ml
```

```
# Демонстрація для одного тікера
demo_ticker = available_tickers[0]
df_ml = prepare_ml_dataset(metrics_data[demo_ticker])

print(f"{demo_ticker} - датасет для ML:")
print(f"  Кількість зразків: {len(df_ml)}")
print(f"  Кількість ознак: {len(df_ml.columns)}")
print(f"  Колонки: {list(df_ml.columns)}")
print(f"\nПерші 5 рядків:")
print(df_ml[['Close', 'Daily_Return', 'Close_Lag_1', 'Target']].head())

# 8.9 МАСШТАБУВАННЯ ДЛЯ ML
print("\n--- 8.9 Масштабування для машинного навчання ---\n")

# Відокремлюємо ознаки та ціль
feature_cols = [col for col in df_ml.columns if col != 'Target']
X = df_ml[feature_cols].values
y = df_ml['Target'].values

# Демонструємо різні методи масштабування
scalers = {
    'StandardScaler': StandardScaler(),
    'MinMaxScaler': MinMaxScaler(),
    'RobustScaler': RobustScaler()
}

for name, scaler in scalers.items():
    X_scaled = scaler.fit_transform(X)
    print(f"{name}:")
    print(f"  Min: {X_scaled.min():.3f}, Max: {X_scaled.max():.3f}")
    print(f"  Mean: {X_scaled.mean():.3f}, Std: {X_scaled.std():.3f}")
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Вивчіть та повторіть наведені приклади.
2. Проаналізуйте якість даних для тікерів свого варіанту. Результат сформуйте у таблицю з метриками якості даних та описами їх аномалій.

варіант	тікери
1	AAPL, HSBA.L, SGR.AX, CAT.AX
2	GOOGL, BP.L, 29M.AX, BWP.AX
3	MSFT, VOD.L, ACF.AX, TUA.AX
4	TSLA, SAP.DE, MYE.AX, RWC.AX
5	AMZN, BMW.DE, ECK.MI, VOW3.DE
6	NVDA, ALV.DE, GUI.PA, FRCB
7	META, AIR.PA, QBY.DE, GEHC
8	GOOGL, OR.PA, ICP1V.HE, BMPS.MI
9	AMZN, 7203.T, MIKN.SW, 0460.HK
10	TSLA, 6758.T, DXB.AX, THG.L

3. Виконайте аналіз базових фінансових метрик для тікерів свого варіанту.
4. Реалізуйте клас виявлення аномалій у фінансових даних за наступною структурою:

```
class AnomalyDetector:
    """
    Реалізуйте клас для виявлення аномалій різними методами
    """

    def __init__(self, df, ticker):
        self.df = df
        self.ticker = ticker

    def z_score_method(self, threshold=3):
        """
```

```
        Виявлення аномалій за Z-оцінкою
        """
        pass

    def rolling_std_method(self, window=20, std_mult=3):
        """
        Виявлення аномалій на основі ковзного стандартного відхилення
        """
        pass

    def compare_methods(self):
        """
        Порівняти результати різних методів
        """
        pass

    def visualize_anomalies(self):
        """
        Візуалізувати знайдені аномалії
        """
        pass

# Приклад тестування
detector = AnomalyDetector(data['AAPL'], 'AAPL')
detector.visualize_anomalies()
```

Продемонструйте використання класу для двох тікерів із свого варіанту.

5. Реалізувати метод порівняльного аналізу нормалізації

```
def compare_normalization_methods(df, methods=['minmax', 'standard', 'robust']):
    """
    1. Застосувати всі методи нормалізації
    2. Порівняти розподіли до/після
    3. Дослідити вплив нормалізації на виявлення аномалій
    """
    pass
```

У своєму варіанті візьміть перший тікер (ліквідний) та один проблемний тікер і порівняйте результати роботи методу.

6. Підготуйте звіт з виконання практичних завдань.