ЛАБОРАТОРНА РОБОТА 3

Тема: Оцінка якості даних

Існують формальні критерії якості даних, на основі яких в професійному середовищі приймається рішення, чи можуть дані використовуватись у бізнес-аналітиці, ML або фінансовому моделюванні. До класичних критеріїв якості даних відносяться:

- Повнота (completeness)
- Точність (accuracy)
- Узгодженість (consistency)
- Актуальність (timeliness)
- Валідність (validity)
- Унікальність (uniqueness)

У прикладі 1 наведено клас, в якому реалізовано методи оцінки якості даних за вказаними критеріями.

Повнота найчастіше враховує наявність пропущених календарних днів у часовому інтервалі. Інколи до цього показника можуть додаватись і торгові дні із нульовим об’ємом. Тобто критерій вимірює не лише відсутність записів, а й наявність "псевдозаповнених" записів (zero volume). Тоді такий критерій стає інтегральним. У дійсності будь-які критерії можуть представляти собою інтегральну величину — декілька складових, вплив яких у загальну величину враховується з певним коефіцієнтом.

Критерій реалізовано в методі assess_completeness(self). Перша та остання дати визначено df.index.min() та df.index.max() відповідно. Параметр freq='D' вказує на всі календарні дні (включно з вихідними). Таким чином, загальна кількість календарних днів calendar_days, а кількість фактичних записів у DataFrame із даними trading_days. Пропущені дні визначаються як долю (або відсоток) за формулою

$$Missing\,\% = \frac{CalendarDays - TradingDays}{CalendarDays}$$

Для торговельних мереж робота у вихідні та святкові дні є звичайною і ця формула в цьому випадку залишається актуальною, але такий режим роботи не має абсолютного застосування. Наприклад, торговельні павільйони на міських ринках не працюють у понеділок та деякі святкові дні, а фондові ринки не торгують у вихідні та святкові дні. Тому цей показник можна зробити більш відповідним до врахування реальних пропусків і виключити із розгляду неторгові дні. Наприклад, параметр freq='B' вказує на «business days» для фондових ринків.

Нульовий об’єм вимірюється як відношення кількості днів з нульовим об’ємом до кількості торгових днів:

$$ZeroVolume\,\% = \frac{DaysWithVolume = 0}{TradingDays}$$

Для ліквідних акцій це зазвичай або ознака проблеми даних, або делістинг, або неактивний актив. Для магазинів це ознака або проблеми даних, або відсутність товару, або інша вагома причина втрати можливості торгівлі.

Інтегральний бал формується за наступною формулою:

$$Completeness = 1 - Missing\,\% - 0.5 \times ZeroVolume\,\%$$

Тобто в інтегральному показнику вплив пропущених днів вищий (коефіцієнт 1.0), ніж днів з нульовим об’ємом торгівлі (коефіцієнт 0.5). Коефіцієнти надають змістовності впливу кожної із складових і часто або обґрунтовуються, або визначаються через експериментальну модель.
Інтерпретація отриманих значень:
0.9–1.0 - висока повнота
0.7–0.9 - прийнятна
0.5–0.7 - проблемна
<0.5 - непридатна для бізнес-аналітики.

В залежності від подальшого використання даних у бізнес-аналітиці якщо completeness низька, то:

- ML-модель може навчатися на спотворених розподілах
- Волатильність буде некоректною
- RSI може бути помилковим
- Backtesting стане ненадійним

Метрика якості за критерієм Accuracy (точність / достовірність значень) виявляє логічні, статистичні та аномальні порушення в цінових даних та перетворює їх у штрафну функцію за лінійною моделлю без вагової нормалізації:

$$Accuracy = max(0, 1 - penalty)$$

Першим перевіряються колонки даних High, Low, Close і значення в них має відповідати відношенню:

$$Low \leq Close \leq High$$

Порушення означає: помилку джерела, пошкодження даних, неправильну агрегацію або split без корекції. Штраф за кожен тип логічного порушення визначається, як: penalty += 0.1.
Далі перевіряються негативні або нульові ціни в колонці Close. Нуль або від’ємне значення означає: серйозну помилку, неправильне парсування або пошкодження даних. Штраф: penalty+=0.2.
Наступним визначаються аномальні денні зміни (>20%), в яких використовується колонка Close та визначаються зміни за допомогою функції pct_change(). Якщо повертаються значення >20%, то підраховуються: extreme_returns та extreme_pct і штраф буде пропорційний частці екстремальних змін: penalty += extreme_pct. Це відповідає моделі із жорстко заданим емпіричним порогом і може не відповідати звичайній ситуації, наприклад, для криптовалют або акцій з малою капіталізацією.
Останнім робиться Z-оцінка або викиди. Для її визначення приймають участь:
Close.mean() - середнє арифметичне значення ціни закриття або "типовий рівень" ціни за період

$$\mu = \frac{1}{N}\sum_{i=1}^{N} Close_i$$

та Close.std() - стандартне відхилення або масштаб коливань навколо середнього

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Close_i - \mu)^2}$$

Z-оцінка визначається за формулою:

$$Z = \frac{Close - Close.mean}{Close.std}$$

Якщо |Z| > 5 , то нараховується штраф: $penalty = \frac{extreme_z}{N} \times 2$, де extreme_z — кількість таких випадків; N загальна кількість даних в Close.

Аналіз даних за критерієм Consistency (часова узгодженість ряду) має показати наскільки регулярно та передбачувано надходять записи у часовому ряді. Для цього аналізується часовий індекс DataFrame df.index. Критерій перевіряє структуру часової осі, а не значення. Обчислюються такі параметри:
1. Інтервали між записами (вимір у днях)

$$interval_i = Date_i - Date_{i-1}$$

2. Нормальність інтервалів
Нормальними вважаються:

- 1 день - звичайний торговий день
- 3 дні — з п'ятниці по понеділок

Це припускає наявність п’ятиденного біржового тижня і припущення відсутності свят (спрощений варіант).

3. Відсоток нормальних інтервалів є основним показником регулярності:

$$\text{consistency\_pct} = \frac{\text{normal\_intervals}}{\text{total\_intervals}} \times 100$$

4. Великі розриви

Розриви > 5 днів вважаються проблемними і для задачі фондового ринку вказують на ймовірність:

- помилки завантаження
- відсутність даних
- делістинг
- зупинку торгів.

Загальна оцінка формується у два етапи

Визначення базової оцінки (діапазон 0–1):

$$consistency = \text{consistency\_pct} / 100$$

Застосування мультиплікативного штрафу за великі пропуски:

$$consistency = consistency \times \left(1 - \frac{\text{large\_gap}}{\text{total\_intervals}}\right)$$

Інтерпретація значення:

0.95–1.0 - часовий ряд регулярний
0.8–0.95 - незначні пропуски
0.6–0.8 - є аномальні розриви
<0.6 - дані нестабільні

Критерій Timeliness (актуальність / оперативність даних) в найпростішому варіанті виглядає як оцінка часової віддаленості останнього запису від поточного моменту:

$$\text{days\_delay} = \text{current\_date} - \text{last\_date}$$

Жодні значення цін або обсягів не аналізуються.

Оцінка формується за ступінчастою (piecewise constant function) моделлю:

$$T(d) = \begin{cases} 1.0 & d \leq 1 \\ 0.8 & 1 < d \leq 5 \\ 0.5 & 5 < d \leq 20 \\ 0.2 & 20 < d \leq 60 \\ 0.0 & d > 60 \end{cases}$$

та інтерпретується наступним чином

days_delay	**Timeliness score**	**Інтерпретація**
≤ 1	1.0	Актуально
≤ 5	0.8	Невелика затримка
≤ 20	0.5	Застарілі
≤ 60	0.2	Критично застарілі
> 60	0.0	Архів

Зазвичай для бізнес-аналітики застосовують наступні вимоги:

Use case	**Вимога до timeliness**
Intraday trading	до 1 дня
Risk management	1-3 дні
Portfolio analytics	< 5 днів
Research	< 20 днів
Historical modeling	timeliness не критичний

Критерій Validity (валідність) вказує на відповідність даних заданим бізнес-правилам та економічній логіці. На відміну від Accuracy перевіряється не статистична аномальність, а нормативні обмеження домену. Для певного тікера формується список порушень (violations) щодо прийнятої моделі аналізу, накопичується штраф (penalty) і обчислюється фінальна оцінка за адитивною rule-based штрафною моделлю:

$$Validity = max(0, 1 - penalty)$$

До порушень, наприклад, може відноситись припущення щодо нормальності розподілу ціни активів, або об'ємів торгів.
Перевірка застосовується до наступних даних.
1. Додатність ціни
Використовується стовпчик Close і перевіряється бізнес-правило: $Close > 0$
Ціна фінансового активу не може бути ≤ 0, у разі його порушення нараховується штраф: +0.2.

2. Невід'ємність обсягу
Використовується стовпчик Volume, перевіряється бізнес-правило: $Volume \geq 0$
Від'ємний обсяг торгів фізично неможливий, у разі його порушення штраф: +0.1.

3. Обмеження волатильності
Використовується стовпчик Close і розраховується історична волатильність:

$$volatility = \sigma_{annual} = \sigma_{daily} \cdot \sqrt{252}$$

де

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

стандартне відхилення, міра розсіювання даних відносно середнього.

Має дотримуватись наступне бізнес-правило: $\sigma \leq 100\%$
Якщо волатильність має більшу величину, то вважається екстремальною і нараховується штраф:

$$penalty \mathrel{+}= min(0.3, volatility/5)$$

Із практичних спостережень вважається, що значення 100% річної волатильності для акцій — дуже висока, але для криптовалют є припустимою і навіть нормальною. Таким чином це правило є доменно-залежним і може корегуватись в залежності від типу об'єкту активу (об'єкту аналізу).

4. Аномально стабільні ціни
Використовуються стовпчики Close та Open. До даних в них використовується бізнес-правило:

$$\frac{NoChangeDays}{TotalDays} \leq 50\%$$

Перевищення цього значення може означати: відсутність торгів, помилкову агрегацію або штучність даних. У разі виявлення нараховується штраф: +0.2.

На відміну від Accuracy критерій Validity вказує на відповідність бізнес-правилам, порушення нормативних обмежень та на економічну адекватність даних. Крім розглянутих параметрів також можуть бути додані і інші, в залежності від задачі аналізу та типів активів (об’єктів аналізу).

Критерій Uniqueness (унікальність) вказує на відсутність дублювання записів на рівні ключа (дати), повного рядка та послідовних значень. На відміну від consistency чи validity аналізується структурна та частково семантичне дублювання даних.
Для визначеного тикера формується список проблем для перевірки в даних (issues), накопичується штраф (penalty) і визначається фінальна оцінка:

$$Uniqueness = max\left(0, 1 - min\left(penalty, 1\right)\right)$$

У цьому випадку штраф також є адитивною величиною та нормується до максимуму 1.
Перевірка застосовується до наступних даних.
1. Дублікати дат (ключа)

До даних типу DataFrame застосовується метод df.index і перевіряється повторюваність часової мітки. Для денного фінансового ряду дата — це первинний ключ. У разі порушення нараховується пропорційний штраф:

$$penalty \mathrel{+}= \frac{\text{duplicate\_dates}}{N}$$

де $N = len\left(df\right)$

2. Повні дублікати рядків
Застосовується для перевірки всіх колонок DataFrame і перевіряється повна ідентичність записів (дата + всі значення). У разі виявлення нараховується штраф:

$$penalty \mathrel{+}= 2 \cdot \frac{\text{duplicate\_rows}}{N}$$

Вага ×2 вказує, що повний дублікат вважається вельми серйозною проблемою.

3. Повторення ціни підряд
Використовується стовпчик Close і перевіряється частка днів, коли ціна не змінилася відносно попереднього дня. Застосовується наступне бізнес-правило: якщо *price_dup_pct > 10%*, то це вважається аномалією і нараховується штраф:

$$penalty \mathrel{+}= \text{price\_dup\_pct}$$

Безумовно повторення ціни не завжди може бути помилкою, наприклад, якщо актив є малоліквідним.
Таким чином виконується оцінка:

$$Uniqueness = 1 - f\left(D_{index}, D_{rows}, D_{price}\right)$$

де штраф:

$$penalty = \frac{D_{index}}{N} + 2 \cdot \frac{D_{rows}}{N} + D_{\text{price\_pct}}$$

У прикладі 2 наведено код використання класу DataQualityAssessor для даних з Yahoo Finance.

Практичний приклад 1

```
class DataQualityAssessor:
    """
    Клас для оцінки якості даних (доповнення до коду лабораторної роботи 2)
    """

    def __init__(self, data_dict):
        """
        Параметри:
         data_dict: словник виду {ticker: DataFrame} де DataFrame має мультиіндексну
                                                                    структуру
                    або звичайний DataFrame з колонками Open, High, Low, Close, Volume
        """
        self.raw_data = data_dict
        self.processed_data = {}
        self.scores = {}

        # Попередня обробка даних
        self._preprocess_data()

    def _preprocess_data(self):
        """
        Адаптація: обробка мультиіндексної структури
        """
        for ticker, df in self.raw_data.items():
            # Створюємо оброблену копію
            processed = pd.DataFrame(index=df.index)

            # Адаптація: отримуємо Series для кожної колонки
            for col in ['Open', 'High', 'Low', 'Close', 'Volume']:
                if col in df.columns:
                    if isinstance(df[col], pd.DataFrame):
                        # Мультиіндексний випадок
                        processed[col] = df[col].iloc[:, 0]
                    else:
                        # Звичайний DataFrame
                        processed[col] = df[col]
                else:
                    # Якщо колонки немає, створюємо з NaN
                    processed[col] = np.nan

            self.processed_data[ticker] = processed

    # 1. ПОВНОТА (Completeness)
    def assess_completeness(self):
        """
        Оцінка повноти даних
        """
        results = {}

        for ticker, df in self.processed_data.items():
            # порівняння торгових і календарних днів
           all_days = pd.date_range(start=df.index.min(),end=df.index.max(),freq='D')
            trading_days = len(df)
            calendar_days = len(all_days)

            # Пропущені дні
            missing_days = calendar_days - trading_days
            missing_pct = (missing_days / calendar_days) * 100
```

```
            # Дні з нульовим об'ємом
            if 'Volume' in df.columns:
                volume_series = df['Volume']
                zero_volume = (volume_series == 0).sum()
                zero_volume_pct = (zero_volume / trading_days) * 100
            else:
                zero_volume = 0
                zero_volume_pct = 0

            # Інтегральна оцінка повноти (0-1)
            # Враховуємо: пропущені дні, нульовий об'єм
            completeness = 1.0 - (missing_pct / 100) - (zero_volume_pct / 100 * 0.5)
            completeness = max(0, min(1, completeness))  # Обмежуємо [0,1]

            results[ticker] = {
                'score': completeness,
                'details': {
                    'trading_days': trading_days,
                    'calendar_days': calendar_days,
                    'missing_days': missing_days,
                    'missing_pct': missing_pct,
                    'zero_volume': zero_volume,
                    'zero_volume_pct': zero_volume_pct
                }
            }

        self.scores['completeness'] = results
        return results

    # 2. ТОЧНІСТЬ (Accuracy)
    def assess_accuracy(self):
        """
        Оцінка точності даних
        """
        results = {}

        for ticker, df in self.processed_data.items():
            issues = []
            penalty = 0

            # 1. Логічні перевірки цінових відношень
            if (df['High'] < df['Low']).any():
                issues.append("High < Low")
                penalty += 0.1

            if (df['High'] < df['Close']).any():
                issues.append("High < Close")
                penalty += 0.1

            if (df['Low'] > df['Close']).any():
                issues.append("Low > Close")
                penalty += 0.1

            # 2. Перевірка на негативні ціни
            if (df['Close'] <= 0).any():
                issues.append("Негативні ціни")
                penalty += 0.2

            # 3. Аномальні зміни ціни
            returns = df['Close'].pct_change() * 100
            extreme_returns = (abs(returns) > 20).sum()
            if extreme_returns > 0:
             extreme_pct = (extreme_returns / len(df)) * 100
             issues.append(f"{extreme_returns} днів зміна >20% ({extreme_pct:.1f}%)")
             penalty += extreme_pct / 100  # Пропорційний штраф

            # 4. Z-оцінка для виявлення викидів
            z_scores = np.abs((df['Close'] - df['Close'].mean()) / df['Close'].std())
            extreme_z = (z_scores > 5).sum()
            if extreme_z > 0:
                issues.append(f"{extreme_z} викидів за Z-оцінкою >5σ")
                penalty += extreme_z / len(df) * 2
```

```
            # Фінальна оцінка
            accuracy = max(0, 1 - penalty)

            results[ticker] = {
                'score': accuracy,
                'details': {
                    'issues': issues,
                    'penalty': penalty,
                          'extreme_returns': extreme_returns if 'extreme_returns' in
locals() else 0,
                    'extreme_z': extreme_z if 'extreme_z' in locals() else 0
                }
            }

        self.scores['accuracy'] = results
        return results

    # 3. УЗГОДЖЕНІСТЬ (Consistency)
    def assess_consistency(self):
        """
        Оцінка узгодженості
        """
        results = {}

        for ticker, df in self.processed_data.items():
            # Аналіз інтервалів між торговими днями
            intervals = df.index.to_series().diff().dt.days
            intervals = intervals.dropna()

            normal_intervals = intervals.isin([1, 3]).sum()
                  consistency_pct = (normal_intervals / len(intervals)) * 100 if
len(intervals) > 0 else 100

            # Великі пропуски (>5 днів)
            large_gaps = intervals[intervals > 5]

            # Оцінка (0-1)
            consistency = consistency_pct / 100
            # Штраф за великі пропуски
            if len(large_gaps) > 0:
                consistency *= (1 - len(large_gaps) / len(intervals))

            results[ticker] = {
                'score': consistency,
                'details': {
                    'min_interval': intervals.min() if len(intervals) > 0 else 0,
                    'max_interval': intervals.max() if len(intervals) > 0 else 0,
                    'avg_interval': intervals.mean() if len(intervals) > 0 else 0,
                    'normal_intervals_pct': consistency_pct,
                    'large_gaps': len(large_gaps),
                                'large_gaps_dates': [d.strftime('%Y-%m-%d') for d in
large_gaps.index[:3]] if len(large_gaps) > 0 else []
                }
            }

        self.scores['consistency'] = results
        return results

    # 4. АКТУАЛЬНІСТЬ (Timeliness)
    def assess_timeliness(self):
        """
        Оцінка актуальності даних
        """
        results = {}
        current_date = datetime.now()

        for ticker, df in self.processed_data.items():
            last_date = df.index.max()
            days_delay = (current_date - last_date).days

            # Оцінка актуальності
```

```
            if days_delay <= 1:
                timeliness = 1.0
                status = "Актуально"
            elif days_delay <= 5:
                timeliness = 0.8
                status = "Невелика затримка"
            elif days_delay <= 20:
                timeliness = 0.5
                status = "Застарілі дані"
            elif days_delay <= 60:
                timeliness = 0.2
                status = "Критично застарілі"
            else:
                timeliness = 0.0
                status = "Архівні дані"

            results[ticker] = {
                'score': timeliness,
                'details': {
                    'last_date': last_date.strftime('%Y-%m-%d'),
                    'days_delay': days_delay,
                    'status': status
                }
            }

        self.scores['timeliness'] = results
        return results

    # 5. ВАЛІДНІСТЬ (Validity)
    def assess_validity(self):
        """
        Оцінка валідності даних (відповідність бізнес-правилам)
        """
        results = {}

        for ticker, df in self.processed_data.items():
            violations = []
            penalty = 0

            # 1. Ціни мають бути додатними
            if (df['Close'] <= 0).any():
                violations.append("Негативні або нульові ціни")
                penalty += 0.2

            # 2. Об'єм має бути невід'ємним
            if 'Volume' in df.columns and (df['Volume'] < 0).any():
                violations.append("Від'ємний об'єм торгів")
                penalty += 0.1

            # 3. Перевірка волатильності (не більше 100% річних)
            returns = df['Close'].pct_change().dropna()
            volatility = returns.std() * np.sqrt(252)
            if volatility > 1.0:  # >100% річна волатильність
                violations.append(f"Екстремальна волатильність: {volatility:.1%}")
                penalty += min(0.3, volatility / 5)

            # 4. Перевірка на стабільність (не більше 50% днів без змін)
            no_change = (df['Close'] == df['Open']).sum()
            no_change_pct = no_change / len(df)
            if no_change_pct > 0.5:
                violations.append(f"Аномально багато днів без змін:
{no_change_pct:.1%}")
                penalty += 0.2

            # Фінальна оцінка
            validity = max(0, 1 - penalty)

            results[ticker] = {
                'score': validity,
                'details': {
                    'violations': violations,
                    'penalty': penalty,
```

```
                    'volatility': volatility if 'volatility' in locals() else 0,
                       'no_change_pct': no_change_pct if 'no_change_pct' in locals()
else 0
                }
            }

        self.scores['validity'] = results
        return results

    # 6. УНІКАЛЬНІСТЬ (Uniqueness)
    def assess_uniqueness(self):
        """
        Оцінка унікальності даних (відсутність дублікатів)
        """
        results = {}

        for ticker, df in self.processed_data.items():
            issues = []
            penalty = 0

            # 1. Дублікати дат
            duplicate_dates = df.index.duplicated().sum()
            if duplicate_dates > 0:
                issues.append(f"{duplicate_dates} дублікатів дат")
                penalty += duplicate_dates / len(df)

            # 2. Повні дублікати рядків
            duplicate_rows = df.duplicated().sum()
            if duplicate_rows > 0:
                issues.append(f"{duplicate_rows} повних дублікатів рядків")
                penalty += duplicate_rows / len(df) * 2

            # 3. Дублікати цін підряд (застій)
            price_duplicates = (df['Close'] == df['Close'].shift(1)).sum()
            price_dup_pct = price_duplicates / len(df)
            if price_dup_pct > 0.1:  # >10% днів з тією ж ціною
                issues.append(f"Аномально багато повторів цін: {price_dup_pct:.1%}")
                penalty += price_dup_pct

            # Фінальна оцінка
            uniqueness = max(0, 1 - min(penalty, 1))

            results[ticker] = {
                'score': uniqueness,
                'details': {
                    'issues': issues,
                    'penalty': penalty,
                    'duplicate_dates': duplicate_dates,
                    'duplicate_rows': duplicate_rows,
                    'price_duplicates_pct': price_dup_pct
                }
            }

        self.scores['uniqueness'] = results
        return results

    def comprehensive_report(self):
        """
        Повний звіт про якість даних
        """
        # Виконуємо всі оцінки
        self.assess_completeness()
        self.assess_accuracy()
        self.assess_consistency()
        self.assess_timeliness()
        self.assess_validity()
        self.assess_uniqueness()

        # Створюємо зведений DataFrame
        report_data = []
        for ticker in self.processed_data.keys():
            row = {'Ticker': ticker}
```

```
            for criterion in ['completeness', 'accuracy', 'consistency',
                              'timeliness', 'validity', 'uniqueness']:
                row[criterion.capitalize()] = self.scores[criterion][ticker]['score']

            # Загальна оцінка
            scores = [row[c.capitalize()] for c in ['completeness', 'accuracy',
'consistency',
                                                    'timeliness', 'validity',
'uniqueness']]
            row['Overall'] = np.mean(scores)

            # Рекомендація
            if row['Overall'] < 0.5:
                row['Recommendation'] = 'НЕ ПРИДАТНІ'
            elif row['Overall'] < 0.7:
                row['Recommendation'] = 'Обмежене використання'
            elif row['Overall'] < 0.9:
                row['Recommendation'] = 'Придатні'
            else:
                row['Recommendation'] = 'ВІДМІННА якість'

            report_data.append(row)

        report_df = pd.DataFrame(report_data)
        report_df = report_df.set_index('Ticker')

        return report_df

    def detailed_report(self, ticker):
        """
        Детальний звіт для конкретного тікера
        """
        if ticker not in self.processed_data:
            return f"Тікер {ticker} не знайдено"

        print(f"\n{'='*60}")
        print(f"ДЕТАЛЬНИЙ ЗВІТ ПРО ЯКІСТЬ ДАНИХ: {ticker}")
        print(f"{'='*60}")

        for criterion in ['completeness', 'accuracy', 'consistency',
                          'timeliness', 'validity', 'uniqueness']:
            data = self.scores[criterion][ticker]
            print(f"\n▸ {criterion.upper()}: {data['score']:.1%}")

            if 'details' in data:
                for key, value in data['details'].items():
                    if isinstance(value, list):
                        if value:
                            print(f"  • {key}: {value}")
                    elif isinstance(value, float):
                        print(f"  • {key}: {value:.2f}")
                    else:
                        print(f"  • {key}: {value}")

    def visualize_quality(self):
        """
        Візуалізація якості даних
        """
        report = self.comprehensive_report()

        fig, axes = plt.subplots(1, 2, figsize=(15, 6))

        # Теплова карта
        criteria = ['Completeness', 'Accuracy', 'Consistency',
                    'Timeliness', 'Validity', 'Uniqueness']
        data_for_heatmap = report[criteria].T

        sns.heatmap(data_for_heatmap, annot=True, fmt='.2f',
                    cmap='RdYlGn', vmin=0, vmax=1,
                    cbar_kws={'label': 'Оцінка якості'},
                    ax=axes[0])
        axes[0].set_title('Матриця якості даних за критеріями')
```

```
        axes[0].set_xlabel('Тікер')
        axes[0].set_ylabel('Критерій')

        # Стовпчикова діаграма загальних оцінок
        bars = axes[1].bar(report.index, report['Overall'])
         axes[1].axhline(y=0.7, color='red', linestyle='--', label='Поріг придатності
(0.7)')
        axes[1].axhline(y=0.5, color='orange', linestyle='--', label='Критичний поріг
(0.5)')
        axes[1].set_title('Загальна оцінка якості даних')
        axes[1].set_xlabel('Тікер')
        axes[1].set_ylabel('Загальна оцінка (0-1)')
        axes[1].legend()
        axes[1].grid(True, alpha=0.3)

        # Фарбуємо стовпчики залежно від оцінки
        for bar, score in zip(bars, report['Overall']):
            if score >= 0.9:
                bar.set_color('darkgreen')
            elif score >= 0.7:
                bar.set_color('lightgreen')
            elif score >= 0.5:
                bar.set_color('orange')
            else:
                bar.set_color('red')

            plt.suptitle('Професійна оцінка якості фінансових даних', fontsize=14,
fontweight='bold')
        plt.tight_layout()
        plt.show()
```

Практичний приклад 2

```
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')

# Ваші тікери
tickers = [
    'AAPL',        #
    'SGR.AX',      #
    '29M.AX',      #
    'ACF.AX',      #
    'MYE.AX',      #
]

# Завантажуємо дані 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}")
# Створюємо асесор з нашими даними
assessor = DataQualityAssessor(data)
```

```
# Отримуємо повний звіт
quality_report = assessor.comprehensive_report()
print("\n=== ЗВЕДЕНИЙ ЗВІТ ПРО ЯКІСТЬ ===\n")
print(quality_report)

# Детальний звіт для конкретного тікера
assessor.detailed_report('SGR.AX')

# Візуалізація
assessor.visualize_quality()

# Прийняття рішення про використання
print("\n=== РІШЕННЯ ПРО ВИКОРИСТАННЯ ===\n")
for ticker, row in quality_report.iterrows():
    if row['Overall'] >= 0.7:
        print(f"✅ {ticker}: МОЖНА використовувати (оцінка: {row['Overall']:.1%})")
    elif row['Overall'] >= 0.5:
        print(f"⚠️ {ticker}: З ОБЕРЕЖНІСТЮ (оцінка: {row['Overall']:.1%})")
        print(f"   Проблеми: {[c for c in ['Completeness', 'Accuracy', 'Consistency',
'Timeliness', 'Validity', 'Uniqueness'] if row[c] < 0.7]}")
    else:
        print(f"❌ {ticker}: НЕ ВИКОРИСТОВУВАТИ (оцінка: {row['Overall']:.1%})")
```

Практичні завдання

1. Вивчіть практичні приклади теоретичної частини на тікерах свого варіанту в лабораторній роботі 2. Отримайте показники якості для кожного тікеру.
2. Доповніть код прикладу 2 таким чином, щоб отримані з Yahoo Finance дані могли штучно погіршувати якість по кожному із наведених критеріїв. Реалізуйте можливість вибору критерію, або використання комплексного впливу (декілька критеріїв обираються випадково).
3. Для одного із тікерів свого варіанту виконайте окремо для кожного із критеріїв штучне погіршення якості даних з наступною її оцінкою методами наведеного в лабораторній роботі класу. Зробіть порівняння оцінки якості первинних даних із результатами після їх штучного погіршення.
4. Виконайте експеримент з комплексним погіршенням даних і також виконайте порівняння з оцінкою якості первинних даних.
5. Для одного тікеру за одним критерієм виконайте дослідження, як різні рівні погіршення (0.1, 0.3, 0.5) впливають на оцінку якості даних. Побудуйте графік залежності «рівень погіршення → оцінка якості». Визначте критичний рівень погіршення за цим критерієм, після якого дані стають непридатними.
6. Порівняйте чутливість різних тікерів до однакового погіршення якості даних за цим критерієм.
7. Підготуйте звіт з виконання практичних завдань.