

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Дніпровський національний університет імені Олеся Гончара
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Дніпровський національний університет імені Олеся Гончара

Кваліфікаційна наукова
праця на правах рукопису

ЗЕМЛЯНИЙ ОЛЕКСІЙ ДМИТРОВИЧ

УДК 004.42

ДИСЕРТАЦІЯ

**РОЗРОБЛЕННЯ МЕТОДІВ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ІНТЕЛЕКТУАЛЬНОГО ІМПУТУВАННЯ ПРОПУСКІВ ДАНИХ**

12 Інформаційні технології

121 Інженерія програмного забезпечення

Подається на здобуття ступеня доктора філософії. Дисертація містить результати власних досліджень. Використання ідей, результатів та текстів інших авторів мають посилання на відповідне джерело

_____ О.Д. Земляний

Науковий керівник:
Байбуз Олег Григорович
доктор технічних наук, професор

Дніпро – 2026

АНОТАЦІЯ

Земляний О.Д. Розроблення методів та програмного забезпечення інтелектуального імпутування пропусків даних. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 121 Інженерія програмного забезпечення – Дніпровський національний університет імені Олеся Гончара, Дніпро, 2026.

Проблема пропусків у даних є актуальною для багатьох прикладних галузей, оскільки вона погіршує якість і надійність аналізу, знижує точність алгоритмів машинного навчання та може призводити до хибних висновків, особливо в задачах класифікації та прогнозування. Це ускладнює створення моделей, які б адекватно описували реальні процеси, і збільшує ймовірність упереджених рішень у таких сферах, як охорона здоров'я, екологічний моніторинг та інші.

Дисертація присвячена розробленню методів, алгоритмів та програмних засобів інтелектуального імпутування пропущених значень, які забезпечують точність та ефективність у задачах моніторингу.

Запропоновані методи поєднують підходи на основі класифікації, регресії, ентропійного аналізу та кореляційних залежностей, враховують структуру даних та патерни пропусків. Методи реалізовано у вигляді програмного забезпечення відповідно до архітектурних принципів бібліотеки `scikit-learn` на мові програмування Python для їх уніфікованого застосування. Ефективність методів перевірено на медичних даних та даних гідрологічного моніторингу, де вони продемонстрували здатність покращувати точність класифікації та прогнозування.

У першому розділі дослідження проводиться аналіз сучасних підходів до обробки даних, зокрема проблеми пропусків, яка є однією з ключових перешкод для ефективного машинного навчання. Проведено огляд методів

аналізу даних при діагностуванні ішемічної хвороби серця як прикладу задачі класифікації, огляд задачі прогнозування рівня та витрат води басейну ріки Дніпро як прикладу аналізу часових рядів, а також детальний огляд існуючих методів імпутування пропусків у даних, їх класифікацію за механізмами виникнення та стратегії заповнення.

Другий розділ присвячено алгоритмічному забезпеченню імпутування пропусків у даних. У ньому описано розроблені методи імпутування, які враховують структуру даних, патерни пропусків, кореляційні та ентропійні зв'язки між ознаками, а також алгоритми перетворення якісних ознак на кількісні зі збереженням інформації про пропуски. У розділі наведено UML-діаграми діяльності для візуалізації логіки роботи алгоритмів.

Третій розділ дисертації присвячений програмному забезпеченню методів імпутування пропусків у даних, зокрема реалізації розроблених алгоритмів на мові Python з інтеграцією в екосистему бібліотеки scikit-learn, а також оптимізації коду для покращення продуктивності обчислень.

Четвертий розділ дисертації присвячений тестуванню розроблених методів імпутування пропусків у даних. У ньому описано датасети для задач класифікації та прогнозування часових рядів, а також проведено чотири типи тестів. Перший тест порівнює точність імпутування та швидкодію при використанні даних із штучно доданими пропусками та моделюванням певних патернів. Другий і третій типи тестів аналізують вплив імпутування на якість моделей у задачах класифікації та прогнозування відповідно. Тест четвертого типу дозволяє проаналізувати вплив імпутування на зменшення умовної ентропії ознак.

У **висновках** представлено результати дослідження. Зазначається, що розроблені методи імпутування пропусків у даних підвищують точність імпутування та покращують якість моделей у задачах класифікації та прогнозування.

Наукова новизна одержаних результатів полягає у наступному:

1. **Удосконалено** метод імпутування NoNa шляхом розробки методу UnifiedClassRegrImputer, який також застосовує комбінацію класифікатора та регресора з урахуванням типу даних ознак, при цьому враховує патерни пропусків у даних та надає змінний порядок обходу ознак залежно від кількості пропусків, що дозволяє підвищити точність імпутування та покращити якість моделей у задачах класифікації та прогнозування.

2. **Вперше** розроблено метод імпутування RegrImputer, який застосовує регресор з уточненням значення у випадку категоріальних даних, що дозволяє значно підвищити швидкодію та точність імпутування, а також покращити якість моделей в задачах класифікації та прогнозування.

3. **Удосконалено** ентропійний метод EntropyImputer шляхом додавання двох модифікацій: однокрокова та ітераційна процедури зменшення умовної ентропії кожної ознаки у послідовному та паралельному виконанні, що дозволяє зменшити умовну ентропію та час виконання алгоритму.

4. **Вперше** розроблено гібридний метод імпутування HybridRegrEntropyImputer, який поєднує ентропійний та регресійний підходи в залежності від типу даних ознаки та патернів пропусків, що дозволяє підвищити точність імпутування та швидкодію у порівнянні з EntropyImputer.

5. **Вперше** розроблено метод імпутування CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками, який автоматизує вибір найкращої моделі залежності серед лінійних та квазілінійних моделей, що дозволяє підвищити точність імпутування, а також точність моделей в задачі прогнозування часових рядів.

6. **Вперше** запропоновано метод імпутування пропусків у часових рядах з сезонною складовою на основі виявленого кореляційного зв'язку між ознаками (метод імпутування CorrelationImputer), але дія виконується для кожного окремого місяця року. Це дозволяє підвищити точність імпутування і якість моделей прогнозування у порівнянні з аналогом без розбиття по місяцям.

7. **Удосконалено** методи перетворення якісних ознак на кількісні шляхом збереження інформації про пропуски та можливістю виконувати зворотне перетворення:

- IgnoreNaNLabelEncoder – модифікація стандартного LabelEncoder, яка не кодує пропуски, зберігає словник та дозволяє зворотне перетворення;
- IgnoreNaNFrequentEncoder – модифікація LabelEncoder та FrequencyEncoder, яка враховує частоту значень, надаючи більшу вагу найпоширенішим або найрідшим категоріям, не кодує пропуски, зберігає словник та дозволяє зворотне перетворення.

Практичне значення роботи полягає у наступному:

1. Розроблено алгоритми та програмне забезпечення для імпутування пропусків у даних, яке реалізує запропоновані методи (UnifiedClassRegrImputer, RegrImputer, EntropyImputer, HybridRegrEntropyImputer, CorrelationImputer, IgnoreNaNLabelEncoder, IgnoreNaNFrequentEncoder). Всі алгоритми написані мовою програмування Python відповідно до архітектурних принципів бібліотеки scikit-learn, що забезпечує їхню уніфіковану інтеграцію у стандартні процеси обробки даних. Це дозволяє використовувати методи в аналітичних конвеєрах (pipeline) разом з імпутуванням, масштабуванням, класифікацією чи прогнозуванням.

2. Розроблені методи дозволяють підвищити точність імпутування пропусків у даних медичного та гідрологічного моніторингу. Ефективність методів за точністю значною мірою залежить від особливостей даних, патернів пропусків та їхньої кількості. Метод RegrImputer виділяється найвищою обчислювальною ефективністю. Метод EntropyImputer забезпечує найбільш значне зменшення умовної ентропії ознак щодо цільової змінної. Метод HybridRegrEntropyImputer демонструє підвищену точність імпутування та обчислювальну ефективність порівняно з EntropyImputer. За наявності кореляційних залежностей між ознаками метод CorrelationImputer забезпечує найвищу точність імпутування. У разі наявності сезонної компоненти в часових рядах та вираженої кореляційної залежності модифікація методу

CorrelationImputer з окремим підбором моделей для кожного місяця року демонструє вищу точність імпутування, однак вимагає більших обчислювальних витрат. Метод UnifiedClassRegrImputer виявляє стабільну ефективність за точністю імпутування при різних налаштуваннях, що залежить від кількості пропусків та патернів у даних.

3. Розроблені методи дозволяють покращити якість моделей класифікації та прогнозування для даних медичного та гідрологічного моніторингу у порівнянні з базовими моделями без імпутування та найпростішим стандартним методом імпутування SimpleImputer.

4. Впроваджено результати дисертації в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара. Розроблені методи та програмне забезпечення використовуються при викладанні дисциплін «Аналіз даних на мові Python», «Оптимізація та підвищення продуктивності програмного коду», «Аналіз та візуалізація даних», «Технології пошуку структури в даних» для здобувачів першого (бакалаврського) рівнів вищої освіти галузі інформаційних технологій.

5. Розроблені методи та програмне забезпечення можуть бути впроваджені в реальні системи аналізу даних, що дозволить підвищити якість моделей класифікації та прогнозування, зокрема для даних медичного та гідрологічного моніторингу.

Ключові слова: імпутування пропусків даних, алгоритми обробки даних, інтелектуальний аналіз даних, статистична обробка даних, регресія та класифікація, прогнозування, машинне навчання, ентропія, кореляційні зв'язки, випадкова величина та часовий ряд, статистика, архітектура програмного забезпечення, інженерія програмного забезпечення, UML-діаграма, оптимізація алгоритмів.

ANNOTATION

Zemlianyi O.D. Development of methods and software for intelligent missing data imputation. – Qualifying scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 121 Software Engineering – Oles Honchar Dnipro National University, Dnipro, 2026.

The problem of missing data is relevant for many applied fields, as it degrades the quality and reliability of analysis, reduces the accuracy of machine learning algorithms, and can lead to incorrect conclusions, especially in classification and prediction tasks. This complicates the creation of models that would adequately describe real-world processes and increases the likelihood of biased decisions in critical areas such as healthcare and environmental monitoring.

The dissertation is dedicated to the development of methods, algorithms, and software tools for intelligent imputation of missing values, which ensure accuracy and efficiency in monitoring tasks. The proposed methods combine approaches based on classification, regression, entropy analysis, and correlation dependencies, taking into account data structure and missing data patterns. These methods are implemented as software following the architectural principles of the scikit-learn library in Python for standardized application. Their effectiveness has been verified on medical and hydrological monitoring data, where they demonstrated the ability to improve classification and prediction accuracy.

The first section of the study conducts an analysis of modern approaches to data processing, with a particular focus on the problem of missing data, which is one of the key obstacles to effective machine learning. It includes a review of data analysis methods for diagnosing coronary heart disease as an example of a classification task, an overview of the problem of predicting water level and flow rates in the Dnipro River basin as an example of time series analysis, as well as a detailed review of existing imputation methods, their classification by mechanisms of occurrence, and filling strategies.

The second section is dedicated to the algorithmic support for imputation of missing data. It describes the developed imputation methods that account for data structure, missing data patterns, and correlation and entropy relationships between features, as well as algorithms for transforming qualitative features into quantitative ones while preserving information about missing values. The section also includes UML activity diagrams to visualize the logic of the algorithms.

The third section is dedicated to the software implementation of the imputation methods, including the realization of the developed algorithms in Python with integration into the scikit-learn library ecosystem, as well as code optimization to improve computational performance.

The fourth section is devoted to testing the developed imputation methods. It describes datasets for classification and time series forecasting tasks and presents four types of tests. The first test compares imputation accuracy and speed using data with artificially introduced missing values following specific patterns. The second and third types of tests analyse the impact of imputation on model quality in classification and prediction tasks, respectively. The fourth type of test allows analysing the impact of imputation on reducing the conditional entropy of features.

The conclusions present the results of the study. It is noted that the developed methods for imputing missing data increase the accuracy of imputation and improve the quality of models in classification and forecasting tasks.

The scientific novelty of the obtained results is as follows:

1. The NoNa imputation method has been enhanced through the development of UnifiedClassRegrImputer, which applies a combination of classifier and regressor while considering feature data types. It accounts for missing data patterns and provides a variable order of feature traversal depending on the number of missing values, thereby improving imputation accuracy and enhancing model quality in classification and prediction tasks.

2. For the first time, the RegrImputer imputation method has been developed, applying a regressor with value refinement for categorical data. This

enables significant improvements in computational speed and imputation accuracy, as well as enhanced model quality in classification and prediction tasks.

3. The entropy-based `EntropyImputer` method has been enhanced by introducing two modifications: single-step and iterative procedures for reducing the conditional entropy of each feature in both sequential and parallel execution, which allows reducing conditional entropy and improving algorithm execution time.

4. For the first time, the hybrid imputation method `HybridRegrEntropyImputer` has been developed, combining entropy and regression approaches depending on the feature data type and missing data patterns. This allows improving imputation accuracy and speed compared to `EntropyImputer`.

5. For the first time, the `CorrelationImputer` imputation method has been developed based on the detected correlation between features. It automates the selection of the best dependency model among linear and quasi-linear models, improving imputation accuracy as well as the accuracy of models in time series prediction tasks.

6. For the first time, a method for imputation of missing values in time series with a seasonal component has been proposed. Based on identified correlation relationships between features, this approach applies the `CorrelationImputer` method separately for each month of the year, improving imputation accuracy and the quality of prediction models compared to the non-segmented approach.

7. Methods for transforming qualitative features into quantitative ones have been enhanced by preserving information about missing values and enabling reverse transformation:

- `IgnoreNaNLabelEncoder` is a modification of the standard `LabelEncoder` that does not encode missing values, preserves the dictionary, and allows reverse transformation;

- `IgnoreNaNFrequentEncoder` is a modification of `LabelEncoder` and `FrequencyEncoder` that accounts for value frequency, assigning greater weight to the most or least common categories, does not encode missing values, preserves the dictionary, and allows reverse transformation.

The practical significance of the work is as follows:

1. Software for missing data imputation has been developed, implementing the proposed methods (*UnifiedClassRegrImputer*, *EntropyImputer*, *HybridRegrEntropyImputer*, *CorrelationImputer*, *RegrImputer*, *IgnoreNANLabelEncoder*, *IgnoreNANFrequentEncoder*). All algorithms are integrated into the scikit-learn library architecture in Python, ensuring their unified integration into standard data preprocessing pipelines. This allows the methods to be used in analytical pipelines alongside imputation, scaling, classification, or forecasting.

2. The developed methods enable improved accuracy of imputation for missing data in medical and hydrological monitoring. The effectiveness of these methods in terms of accuracy largely depends on data characteristics, missing data patterns, and their quantity. The *RegrImputer* method stands out with the highest computational efficiency. The *EntropyImputer* method provides the most significant reduction in conditional entropy of features relative to the target variable. The *HybridRegrEntropyImputer* method demonstrates improved imputation accuracy and computational efficiency compared to *EntropyImputer*. In the presence of correlation dependencies between features, the *CorrelationImputer* method ensures the highest imputation accuracy. For time series with seasonal components and pronounced correlation dependencies, a modification of the *CorrelationImputer* method with separate model selection for each month of the year demonstrates higher imputation accuracy, although it requires greater computational costs. The *UnifiedClassRegrImputer* method shows stable effectiveness in terms of imputation accuracy across various settings, depending on the number of missing values and patterns in the data.

3. The developed methods enable improving the quality of classification and prediction models for medical and hydrological monitoring data compared to baseline models without imputation and the simplest standard imputation method, *SimpleImputer*.

4. The results of the dissertation have been implemented in the educational process at the Department of Software Engineering and Information Technology at Oles Honchar Dnipro National University. The developed methods and software are used in teaching the disciplines «Data Analysis in Python», «Optimization and Improvement of the Productivity of the Software Code», «Data Analysis and Visualization», «Technologies for Finding Structure in Data» for students of the first (bachelor's) level of higher education in the field of information technologies.

5. The developed methods and software can be implemented in real data analysis systems, enabling improved quality of classification and prediction models, particularly for medical and hydrological monitoring data.

Keywords: imputation of missing data, data processing algorithms, intelligent data analysis, statistical data processing, regression and classification, forecasting, machine learning, entropy, correlation relationships, random variable and time series, statistics, software architecture, software engineering, UML diagram, algorithm optimization.

Список опублікованих праць за темою дисертації

Статті у наукових фахових виданнях України:

1. Земляний О.Д., Байбуз О.Г. Огляд методів інтелектуального аналізу даних та методів машинного навчання при прогнозуванні ішемічної хвороби серця. *Актуальні проблеми автоматизації та інформаційних технологій*, т. 27, Дніпро, 2023, с. 109–129. DOI: <http://dx.doi.org/10.15421/432311> (особистий внесок Земляного О.Д.: проведення аналізу літературних джерел щодо застосування методів машинного навчання для діагностики ішемічної хвороби серця, систематизація існуючих підходів та алгоритмів, а також підготовка порівняльної характеристики наборів даних, аналіз результатів; Байбуза О.Г.: постановка завдання дослідження, узагальнення отриманих результатів).

2. Земляний О.Д., Байбуз О.Г. Методи імпутування пропусків у даних про ішемічну хворобу серця. *Системні технології. Регіональний міжвузівський збірник наукових праць*, вип. 2(151), Дніпро, 2024, с. 33–49. DOI: <https://doi.org/10.34185/1562-9945-2-151-2024-04> (особистий внесок Земляного О.Д.: розроблення та реалізація алгоритмів імпутування пропусків у медичних даних на основі класифікації та регресії для роботи з категоріальними та кількісними ознаками, проведення експериментального тестування запропонованих методів на двох популярних датасетах, оцінка їхньої точності та швидкодії, аналіз впливу імпутування на якість моделей класифікації, програмування алгоритмів, обробка даних, візуалізація та аналіз результатів; Байбуза О.Г.: постановка мети дослідження, контроль та узагальнення отриманих результатів).

3. Земляний О.Д., Байбуз О.Г. Алгоритми імпутування пропусків у даних на основі ентропії. *Системні технології. Регіональний міжвузівський збірник наукових праць*, вип. 6(155), Дніпро, 2024, с. 133–149. DOI: <https://doi.org/10.34185/1562-9945-6-155-2024-12> (особистий внесок Земляного О.Д.: розроблення алгоритму *EntropyImputer* на основі

ентронійного підходу до імпутування пропусків, реалізація методу мовою Python, оптимізація коду для підвищення продуктивності, проведення аналізу ефективності запропонованого методу; Байбуза О.Г.: постановка мети та завдання дослідження, контроль та узагальнення отриманих результатів).

4. Земляний О.Д., Байбуз О.Г. Імпутування пропусків у даних гідрологічного моніторингу. *Актуальні проблеми автоматизації та інформаційних технологій*, т. 28, Дніпро, 2024, с. 147–160. DOI: <http://dx.doi.org/10.15421/432413> (особистий внесок Земляного О.Д.: проведення статистичного аналізу та дослідження особливостей даних гідрологічного моніторингу басейну ріки Дніпро, розроблення методів імпутування, адаптованих для часових рядів, реалізація алгоритмів мовою Python, аналіз можливостей щодо оптимізації обчислень, проведення порівняльного аналізу ефективності розроблених методів порівняно з традиційними методами, аналіз результатів; Байбуза О.Г.: постановка задачі, аналіз результатів).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Oleksii Zemlianyi, Oleh Baibuz. Development of an intelligent monitoring system for making decisions about the state of the cardiovascular system. *Матеріали Міжнародної науково-практичної інтернет-конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*, вип. 95, Переяслав, 2023, с. 62–63. URL: <https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1> (особистий внесок Земляного О.Д.: розроблення алгоритмів послідовного аналізу для прийняття рішень щодо категорії захворювання серцево-судинної системи на основі методів перевірки гіпотез про параметри нормального розподілу, реалізація програмного забезпечення для аналізу даних добового моніторингу артеріального тиску, проведення

порівняльного аналізу класичних та послідовних методів; Байбуза О.Г.: постановка завдання, узагальнення отриманих результатів).

6. Земляний О.Д., Байбуз О.Г. Аналіз існуючих методів інтелектуального аналізу даних при прогнозуванні ішемічної хвороби серця. *Тези доповідей XXI Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2023)»*, Дніпро, 22-24 листопада 2023 р., Дніпро, 2023, с. 133–134. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf> (особистий внесок Земляного О.Д.: проведення огляду літератури щодо застосування методів машинного навчання для діагностики ішемічної хвороби серця, систематизація та порівняльний аналіз алгоритмів, аналіз програмних засобів для обробки даних, підготовка порівняльної характеристики наборів даних, аналіз результатів; Байбуза О.Г.: постановка мети і узагальнення отриманих результатів).

7. Земляний О.Д., Байбуз О.Г. Розроблення методів для імпутування пропусків у даних в архітектурі Scikit-learn Python. *Тези доповідей XXII Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2024)»*, Дніпро, 20-22 листопада 2024 р, Дніпро, 2024, с. 141–143. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf> (особистий внесок Земляного О.Д.: розроблення та реалізація класів для імпутування пропусків згідно архітектурних принципів бібліотеки scikit-learn, забезпечення їхньої сумісності з pipeline, тестування на реальних наборах даних, оцінка точності та швидкодії методів, програмування алгоритмів та обробка даних; Байбуза О.Г.: постановка мети і завдання дослідження, контроль результатів).

8. Земляний О.Д., Байбуз О.Г. Розроблення гібридного методу для імпутування пропусків у даних на основі регресії та ентропійного підходів. *Тези доповідей XXVII Міжнародної конференції «Автоматика 2024»*, Дніпро, 20-22 листопада 2024 р., Дніпро, 2024, с.114. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B>

[0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf](#) (особистий внесок Земляного О.Д.: розроблення гібридного методу *HybridRegrEntropyImputer*, який поєднує регресійний та ентропійний підходи для імпутування кількісних і якісних ознак, реалізація класу мовою Python згідно архітектурних принципів бібліотеки *scikit-learn*, проведення порівняльного аналізу ефективності методу з іншими розробленими імпутерами, оцінка якості імпутування, швидкодії та впливу на зменшення умовної ентропії ознак, аналіз результатів; Байбуза О.Г.: постановка задачі дослідження, аналіз і узагальнення результатів).

9. Земляний О.Д., Байбуз О.Г. Підвищення якості моделей класифікації та прогнозування шляхом імпутації пропусків. *Тези доповідей XXIII Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2025)»*, Дніпро, 19-21 листопада 2025 р., Дніпро: ДНУ, 2025, с.152–154.

URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf> (особистий внесок Земляного О.Д.: розроблення та реалізація методів імпутування пропусків у даних, аналіз ефективності розроблених методів, інтеграція методів в екосистему бібліотеки *scikit-learn*, проведення апробації на реальних наборах даних, аналіз впливу імпутування на якість моделей класифікації та прогнозування, візуалізація та аналіз результатів; Байбуза О.Г.: постановка мети і завдання дослідження, аналіз і узагальнення отриманих результатів).

Наукові праці в інших виданнях, які додатково відображають зміст дисертації:

10. Земляний О.Д., Байбуз О.Г. Порівняння багатопроцесорної та багатопоточної реалізацій ентропійного підходу для імпутування пропусків у

даних на мові програмування Python. *Виклики та проблеми сучасної науки*, т. 2, Дніпро, 2024, с. 300–304. DOI: <https://doi.org/10.15421/cims.2> URL: <https://cims.fti.dp.ua/j/article/view/131/159> (особистий внесок Земляного О.Д.: дослідження підходів щодо оптимізації обчислень при реалізації ентропійного підходу для імпутування пропусків у даних на мові Python, проведення програмного експерименту на реальних наборах даних, дослідження впливу *GIL* на багатопоточну обробку в Python, аналіз результатів; Байбуза О.Г.: постановка задачі, аналіз результатів).

11. Земляний О.Д., Байбуз О.Г. Векторизація обчислень для оптимізації коду на мові програмування Python. *Виклики та проблеми сучасної науки*, т. 3, Дніпро, 2024, с. 144–149. DOI: <https://doi.org/10.15421/cims.3> URL: <https://cims.fti.dp.ua/j/article/view/215/208>. PURL: <https://purl.org/cims/2403.017> (особистий внесок Земляного О.Д.: дослідження застосування векторизації для підвищення продуктивності та читабельності Python-коду, проведення експериментів на реальних наборах даних, аналіз результатів; Байбуза О.Г.: постановка мети дослідження, аналіз результатів).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	19
ВСТУП	21
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ.....	27
1.1 Огляд методів аналізу даних при діагностуванні ішемічної хвороби серця	29
1.2 Огляд задачі прогнозування рівня та витрат води басейну ріки Дніпро34	
1.3 Огляд методів імпутування пропусків у даних.....	39
1.4 Висновок до розділу 1	47
РОЗДІЛ 2. АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ.....	48
2.1 Методи перетворення якісних ознак на кількісні.....	48
2.2 Метод імпутування пропусків у даних UnifiedClassRegrImputer та його модифікації.....	51
2.3 Метод імпутування пропусків у даних RegrImputer.....	58
2.4 Метод імпутування пропусків у даних EntropyImputer	60
2.5 Гібридний метод імпутування HybridRegrEntropyImputer	67
2.6 Метод імпутування CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками	67
2.7 Висновок до розділу 2	71
РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МЕТОДІВ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ.....	72
3.1 Розроблення методів для імпутування пропусків у даних відповідно до архітектурних принципів бібліотеки scikit-learn Python	72
3.2 Векторизація обчислень для оптимізації коду на мові програмування Python.....	76
3.3 Порівняння багатопроцесної та багатопоточної реалізацій ентропійного методу для імпутації пропусків у Python.....	82

3.4	Висновок до розділу 3	84
РОЗДІЛ 4. РЕЗУЛЬТАТИ РОБОТИ МЕТОДІВ ІМПУТУВАННЯ		
ПРОПУСКІВ У ДАНИХ.....		
		86
4.1	Опис датасетів для задач класифікації.....	86
4.2	Опис датасету для задачі прогнозування часових рядів	87
4.3	Аналіз точності імпутування та швидкодії (тест типу 1).....	90
4.4	Аналіз впливу імпутування на точність моделі класифікації (тест типу 2)	99
4.5	Аналіз впливу імпутування на точність моделі прогнозування (тест типу 3)	101
4.6	Аналіз змін ентропії до та після імпутування (тест типу 4).....	105
4.7	Демонстрація роботи методів у конвеєрі (pipeline).....	108
4.8	Висновок до розділу 4	111
ВИСНОВКИ.....		114
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ		118
ДОДАТКИ		132
Додаток А Список праць здобувача за темою дисертації.....		132
Додаток Б Коди класів імпульсів та кодувальників якісних ознак		137
Додаток В Акт впровадження результатів роботи		168

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ССЗ – серцево-судинні захворювання;

ІХС – ішемічна хвороба серця;

ШІ – штучний інтелект;

MAR – Missing At Random, відсутній випадково;

MCAR – Missing Completely At Random, відсутній абсолютно випадково;

MNAR – Missing Not At Random, відсутній не випадково;

UnifiedCRSortedAsc/Desc – метод UnifiedClassRegrImputer із сортуванням за кількістю пропусків у зростаючому/спадному порядку, без реакції на патерн;

UnifiedCR2steps – імплімент UnifiedClassRegrImputer із спадним сортуванням ознак та реакцією на патерн;

RegrValue – імплімент RegrImputer із спадним сортуванням ознак, з реакцією на патерн у даних та з корегуванням значення для категоріальної ознаки за найближчим значенням зі словника;

RegrWeight – імплімент RegrImputer із спадним сортуванням ознак, з реакцією на патерн у даних та з корегуванням значення для категоріальної ознаки за найближчим за середньозваженою оцінкою значення зі словника;

Entropy1/EntropyIter – імплімент EntropyImputer на основі ентропійного підходу, одно- та багатокроковий;

EntropyIterThreads/Processes – імплімент EntropyImputer на основі ентропійного підходу, з використанням потоків чи процесів;

HybridRegrEntropy – гібридний імплімент HybridRegrEntropyImputer на основі ентропійного та регресійного підходів;

Corr – імплімент CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками, виконаний послідовно;

CorrThreads – імплімент CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками, виконаний з використанням потоків;

`CorrProcesses` – імп'ютер `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками, виконаний з використанням процесів;

`CorrMonth` – імп'ютер `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками, виконаний для часових рядів помісячно;

`NoNa` – імп'ютер `NoNa`;

`SimpleImputer`, `SimpleMeanImputer` – імп'ютер `SimpleImputer` з бібліотеки `scikit-learn`, що замінює пропуски середнім значенням;

`SimpleFreqImputer` – імп'ютер `SimpleImputer` з бібліотеки `scikit-learn`, що замінює пропуски найбільш частим значенням;

`IterativeImputer` – імп'ютер `IterativeImputer` з бібліотеки `scikit-learn` на основі MICE-методу множинного імпутування;

`kNNImputer` – імп'ютер `kNN`, k найближчих сусідів ($k = 15$ за замовчуванням) з бібліотеки `scikit-learn`.

ВСТУП

Обґрунтування вибору теми дослідження. Сучасна цифровізація різних сфер життя призводить до зростання обсягів даних, які використовуються для автоматизованого прийняття рішень у медицині, економіці, екологічному моніторингу тощо. Проте під час збирання та обробки даних часто виникають проблеми, пов'язані з їх неповнотою. Причини пропусків у даних можуть бути різними: помилки введення, збої в системах збору або об'єднання даних з різних джерел, які мають неоднорідну структуру. У результаті дані містять пропуски, що можуть значно впливати на точність і надійність аналітичних моделей.

Якщо просто ігнорувати ці пропуски або відкидати спостереження з відсутніми значеннями, це може призвести до упередженості та втрати цінної інформації, що знижує ефективність прийняття рішень. Проблема пропусків є добре відомою, і для її вирішення запропоновано чимало методів імпутації, проте класичні підходи часто не враховують складних взаємозв'язків у даних і можуть бути неефективними при роботі з великими та різнорідними наборами.

Актуальність розробки нових методів імпутації полягає у створенні точніших, адаптивних і продуктивних підходів, які б забезпечували надійність моделей для прийняття рішень навіть у випадку неповних даних. Це дозволить більш ефективно використовувати дані з різних джерел, що важливо для сучасних автоматизованих систем.

Мета і завдання дослідження. Метою дослідження є розроблення методів, алгоритмів та програмних засобів інтелектуального імпутування пропущених значень, які забезпечують точність та ефективність у задачах моніторингу.

Під *інтелектуальним імпутуванням пропусків у даних* ми розуміємо метод заповнення пропусків у даних, який використовує алгоритми машинного навчання, статистичний аналіз і залежності між ознаками для урахування

структури даних та патернів пропусків, щоб покращити якість подальшого аналізу.

Основні завдання дослідження:

1. Провести аналіз існуючих методів імпутування та виявити переваги та обмеження наявних підходів.
2. Розробити нові методи та алгоритми імпутування, а саме запропонувати нові підходи, орієнтовані на різні типи даних і завдань, зокрема методи на основі класифікації та регресії, ентропійному підході та гібридні методи, що поєднують ентропійний і регресійний підходи.
3. Розробити алгоритми та програмне забезпечення методів імпутування на мові програмування Python відповідно до архітектурних принципів бібліотеки scikit-learn.
4. Оцінити ефективність нових методів за точністю імпутування та часом обробки на датасетах медичного та гідрологічного моніторингу.
5. Проаналізувати вплив нових методів імпутування на якість моделей класифікації та прогнозування для даних медичного та гідрологічного моніторингу.

Об'єктом дослідження є процеси обробки, аналізу та підготовки даних в системах моніторингу.

Предметом дослідження є методи алгоритми та програмні засоби інтелектуального імпутування пропущених значень у вибірках для підвищення ефективності обробки даних в системах моніторингу.

Методи дослідження: методи та технології інженерії програмного забезпечення, статистичний аналіз, паралельне програмування, математичне моделювання, імпутування, інформаційна ентропія.

Наукова новизна одержаних результатів полягає у наступному:

1. **Удосконалено** метод імпутування NoNa шляхом розробки методу UnifiedClassRegrImputer, який також застосовує комбінацію класифікатора та регресора з урахуванням типу даних ознак, при цьому враховує патерни пропусків у даних та надає змінний порядок обходу ознак залежно від кількості

пропусків, що дозволяє підвищити точність імпутування та покращити якість моделей у задачах класифікації та прогнозування.

2. **Вперше** розроблено метод імпутування `RegrImputer`, який застосовує регресор з уточненням значення у випадку категоріальних даних, що дозволяє значно підвищити швидкодію та точність імпутування, а також покращити якість моделей в задачах класифікації та прогнозування.

3. **Удосконалено** ентропійний метод `EntropyImputer` шляхом додавання двох модифікацій: однокрокова та ітераційна процедури зменшення умовної ентропії кожної ознаки у послідовному та паралельному виконанні, що дозволяє зменшити умовну ентропію та час виконання алгоритму.

4. **Вперше** розроблено гібридний метод імпутування `HybridRegrEntropyImputer`, який поєднує ентропійний та регресійний підходи в залежності від типу даних ознаки та патернів пропусків, що дозволяє підвищити точність імпутування та швидкодію у порівнянні з `EntropyImputer`.

5. **Вперше** розроблено метод імпутування `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками, який автоматизує вибір найкращої моделі залежності серед лінійних та квазілінійних моделей, що дозволяє підвищити точність імпутування, а також точність моделей в задачі прогнозування часових рядів.

6. **Вперше** запропоновано метод імпутування пропусків у часових рядах з сезонною складовою на основі виявленого кореляційного зв'язку між ознаками (метод імпутування `CorrelationImputer`), але дія виконується для кожного окремого місяця року. Це дозволяє підвищити точність імпутування і якість моделей прогнозування у порівнянні з аналогом без розбиття по місяцям.

7. **Удосконалено** методи перетворення якісних ознак на кількісні шляхом збереження інформації про пропуски та можливістю виконувати зворотне перетворення:

– `IgnoreNaNLabelEncoder` – модифікація стандартного `LabelEncoder`, яка не кодує пропуски, зберігає словник та дозволяє зворотне перетворення;

– IgnoreNANFrequentEncoder – модифікація LabelEncoder та FrequencyEncoder, яка враховує частоту значень, надаючи більшу вагу найпоширенішим або найрідшим категоріям, не кодує пропуски, зберігає словник та дозволяє зворотне перетворення.

Практичне значення роботи полягає у наступному:

1. Розроблено алгоритми та програмне забезпечення для імпутування пропусків у даних, яке реалізує запропоновані методи (UnifiedClassRegrImputer, RegrImputer, EntropyImputer, HybridRegrEntropyImputer, CorrelationImputer, IgnoreNANLabelEncoder, IgnoreNANFrequentEncoder). Всі алгоритми написані мовою програмування Python відповідно до архітектурних принципів бібліотеки scikit-learn, що забезпечує їхню уніфіковану інтеграцію у стандартні процеси обробки даних. Це дозволяє використовувати методи в аналітичних конвеєрах (pipeline) разом з імпутуванням, масштабуванням, класифікацією чи прогнозуванням.

2. Розроблені методи дозволяють підвищити точність імпутування пропусків у даних медичного та гідрологічного моніторингу. Ефективність методів за точністю значною мірою залежить від особливостей даних, патернів пропусків та їхньої кількості. Метод RegrImputer виділяється найвищою обчислювальною ефективністю. Метод EntropyImputer забезпечує найбільш значне зменшення умовної ентропії ознак щодо цільової змінної. Метод HybridRegrEntropyImputer демонструє підвищену точність імпутування та обчислювальну ефективність порівняно з EntropyImputer. За наявності кореляційних залежностей між ознаками метод CorrelationImputer забезпечує найвищу точність імпутування. У разі наявності сезонної компоненти в часових рядах та вираженої кореляційної залежності модифікація методу CorrelationImputer з окремим підбором моделей для кожного місяця року демонструє вищу точність імпутування, однак вимагає більших обчислювальних витрат. Метод UnifiedClassRegrImputer виявляє стабільну ефективність за точністю імпутування при різних налаштуваннях, що залежить від кількості пропусків та патернів у даних.

3. Розроблені методи дозволяють покращити якість моделей класифікації та прогнозування для даних медичного та гідрологічного моніторингу у порівнянні з базовими моделями без імпутування та найпростішим стандартним методом імпутуванням SimpleImputer.

4. Впроваджено результати дисертації в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара. Розроблені методи та програмне забезпечення використовуються при викладанні дисциплін «Аналіз даних на мові Python», «Оптимізація та підвищення продуктивності програмного коду», «Аналіз та візуалізація даних», «Технології пошуку структури в даних» для здобувачів першого (бакалаврського) рівня вищої освіти галузі інформаційних технологій.

5. Розроблені методи та програмне забезпечення можуть бути впроваджені в реальні системи аналізу даних, що дозволить підвищити якість моделей класифікації та прогнозування, зокрема для даних медичного та гідрологічного моніторингу.

Особистий внесок здобувача. Аналіз літературних джерел, розроблення алгоритмів та програмного забезпечення імпутування пропусків у даних, порівняльний аналіз методів, оптимізація коду на мові програмування Python, обробка отриманих результатів виконані автором самостійно. Постановка мети і завдань дослідження, аналіз і узагальнення отриманих результатів проводились спільно з науковим керівником доктором технічних наук, професором О. Г. Байбузом.

Апробація результатів дисертації. Основні положення та результати дисертаційної роботи доповідалися й обговорювалися на XXI, XXII, XXIII Міжнародних науково-практичних конференціях «Математичне та програмне забезпечення інтелектуальних систем (MPZIS)» (м. Дніпро, 2023, 2024, 2025), XXVII Міжнародній конференції «Автоматика-2024» (м. Дніпро, 2024), Виклики та проблеми сучасної науки (м. Дніпро, 2024), Міжнародній науково-практичній інтернет-конференції «Тенденції та перспективи розвитку науки і

освіти в умовах глобалізації» (м. Переяслав, 2023), підсумкових наукових конференціях Дніпровського національного університету імені Олеся Гончара (м. Дніпро, 2023, 2024, 2025, 2026).

Публікації. Основні положення й результати дисертаційної роботи опубліковано у 11 роботах: 4 статті у наукових фахових виданнях України категорії Б, 2 статті у інших збірниках наукових праць та 5 тез доповідей у збірниках матеріалів наукових конференцій.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась у відповідності з індивідуальним планом підготовки аспіранта кафедри інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара. Дослідження здійснювалось в рамках науково-дослідної роботи № ФПМ-2-22 «Розроблення програмного забезпечення аналізу та кластеризації часових рядів» 2022-2024 рр. номер держреєстрації 0122U001465 та № 58 – ФПМ-2-25 «Розроблення інформаційної технології обробки статистичних даних» 2025-27рр. номер держреєстрації 0125U002280.

Дослідження було частково профінансоване Міністерством закордонних справ Чеської Республіки в межах проєкту № 24-PKVV-UM-011 «Посилення стандартів викладання, досліджень та міжнародного співробітництва в Дніпровському національному університеті імені Олеся Гончара (ДНУ)», реалізованого Карловим університетом і ДНУ: Project «Strengthening the Standards of Teaching, Research and International Cooperation at Oles Honchar Dnipro National University (DNU)» granted by the Ministry of Foreign Affairs of the Czech Republic 2024, Registration Number 24-PKVV-UM-011, Activity 2.1.4: Providing mini-grants for research and publication activities of DNU researchers, project name – Imputation of gaps in hydrological monitoring data.

Структура та обсяг дисертації. Робота містить вступ, 4 розділи, висновки, список використаних джерел зі 121 найменування, додатки. Загальний обсяг дисертації – 169 сторінок, обсяг основного тексту – 131 сторінка. Робота містить 17 рисунків та 19 таблиць.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ

Успішне вирішення задач машинного навчання потребує розуміння специфіки предметної області та участі експертів у її обговоренні. Підготовка даних є ключовим етапом, що забезпечує створення якісних наборів для навчання моделей. Навіть за наявності великого обсягу даних і знань про їх аналіз, недостатнє розуміння суті цих даних може зробити навчання моделей неефективним або навіть шкідливим.

Оскільки всі датасети мають певні недоліки, підготовка даних є важливою частиною процесу. Цей етап включає заходи для покращення якості даних і створення надійної системи їх збору. Підготовка може займати значний час – інколи до кількох місяців перед початком навчання моделей [1]. За відсутності спеціалістів із науки про дані ця задача стає складнішою, особливо через специфічність організаційних процесів. Хоча найкращим варіантом є залучення досвідченого фахівця, якісна попередня обробка може спростити завдання.

Розглянемо типові задачі підготовки даних:

1. Робота з відсутніми даними (handling missing data). Пропуски виникають через неповноту або помилки у збиранні інформації. Вони обробляються шляхом імпутації (заповнення) за допомогою середніх значень, медіани, передбачуваних значень на основі інших ознак або алгоритмів машинного навчання.

2. Обробка викидів (outliers). Викиди – це аномальні значення, які можуть бути спричинені вимірювальними помилками або відображати реальні аномалії. Для їх виявлення застосовують статистичні методи або методи машинного навчання.

3. Обробка категоріальних ознак (handling categorical data). Категоріальні дані потрібно трансформувати у кількісні, щоб алгоритми могли їх обробити.

Використовуються методи, такі як label encoding (кодування у вигляді чисел) або one-hot encoding (створення бінарних стовпців для кожної категорії).

4. Обробка зміщених даних (skewed data). Якщо розподіл значень ознаки є асиметричним, це може вплинути на моделі, що передбачають нормальний розподіл. Корекція виконується за допомогою логарифмування або інших математичних перетворень.

5. Фільтрація шуму в даних (noisy data). Причинами шуму в даних можуть бути помилки вимірювань або зовнішні фактори. Обробка шуму включає фільтрацію, видалення аномалій або використання алгоритмів, стійких до шуму.

6. Обробка незбалансованих даних (data imbalance). У задачах класифікації, коли кількість екземплярів одного класу значно перебільшує кількість іншого, моделі можуть бути упередженими. Для вирішення цієї проблеми застосовують методи підвибірки, надвибірки (додавання штучних даних для менш представленого класу) або алгоритми, які враховують вагу класів.

7. Масштабування ознак (feature scaling). В алгоритмах, які працюють з відстанями (наприклад, SVM або kNN), процедура приведення значень ознак до єдиного діапазону дуже важлива. Масштабування, наприклад, мінімаксне масштабування або стандартизація, допомагають уникнути домінування ознак із більшими значеннями.

8. Нормалізація даних (data normalization). Це процес приведення даних до спільної шкали, наприклад, шляхом Z-нормалізації, для покращення збіжності алгоритмів.

9. Інженерія ознак (feature engineering). Це процес створення нових ознак або трансформації існуючих для підвищення прогнозної здатності моделей.

10. Зменшення розмірності (dimensionality reduction). Використовується для усунення надмірних ознак, які можуть ускладнювати модель. Існують методи, які допомагають зменшити кількість ознак, зберігаючи основну інформацію, наприклад, метод головних компонент PCA.

Кожен набір даних має свої унікальні проблеми, і важливо застосовувати відповідні методи залежно від конкретної ситуації. Розглянемо далі дві типові задачі інтелектуального аналізу даних:

1. Класифікація – розглядатиметься на прикладі діагностування ішемічної хвороби серця.

2. Прогнозування часових рядів – наприклад, за даними про рівень води та витрати води в басейні ріки Дніпро.

Обробка даних для цих задач потребує ретельного вирішення проблем із пропущеними значеннями та перетворенням категоріальних ознак.

Огляд методів аналізу даних при діагностуванні ішемічної хвороби серця наведено у підрозділі 1.1. Огляд задачі прогнозування показників про рівень води та витрати води басейну ріки Дніпро представлено у підрозділі 1.2. Підрозділ 1.3 присвячений огляду методів імпутування пропусків у даних. Висновок до розділу міститься у підрозділі 1.4.

1.1 Огляд методів аналізу даних при діагностуванні ішемічної хвороби серця

У дослідженні [102] було здійснено детальний аналіз методів обробки даних, які використовуються для діагностування ішемічної хвороби серця (ІХС). Зокрема, розглянуто використання машинного навчання та інтелектуального аналізу даних, які все частіше згадуються у науковій літературі. Найбільш ґрунтовні огляди з цієї тематики представлені в роботах [2, 3].

За статистикою ВООЗ, серцево-судинні захворювання (ССЗ) щороку стають причиною приблизно 18 мільйонів смертей у світі, і значну частину цих захворювань становлять патології серця та судин [4]. Серед них ішемічна хвороба серця є найбільш поширеною, охоплюючи близько 64% усіх випадків ССЗ [5]. ІХС частіше виявляється у чоловіків, починаючи з четвертого десятиліття життя, і прогресує з віком [6]. Хоча ангиографія вважається золотим стандартом у діагностиці ІХС, цей метод є інвазивним і дорогим [7].

У світлі глобальної значущості ССЗ та високих рівнів смертності важливо забезпечити ранню діагностику та прогнозування захворювань.

Штучний інтелект (ШІ) суттєво сприяє визначенню ключових показників ССЗ, аналізу анамнезу пацієнтів та вибору оптимального лікування. Серед переваг, відзначених у науковій літературі, виділяють оперативність прийняття рішень, точність візуалізації серцево-судинної системи, зниження ризиків при лікуванні, покращення обізнаності лікарів про стан пацієнтів та вдосконалення комп'ютерної діагностики [8]. Алгоритми машинного навчання забезпечують швидку та недорогу діагностику ІХС, демонструючи високу точність, чутливість і специфічність. Ці алгоритми дозволяють аналізувати та інтерпретувати кореляції між змінними, сприяючи підвищенню якості лікування [9].

Машинне навчання поділяється на три основні типи: навчання з учителем, без учителя та з підкріпленням [10]. Навчання з учителем ґрунтується на маркованих даних, де відомі результати використовуються для класифікації нових даних [11]. Неконтрольоване навчання фокусується на знаходженні прихованих структур у немаркованих наборах даних [12]. Навчання з підкріпленням використовує підхід до максимізації винагороди шляхом взаємодії програми із середовищем.

Цей огляд спрямований на визначення найефективніших алгоритмів машинного навчання для діагностики ІХС. Критерії відбору публікацій включали ключові слова, пов'язані з кардіологією, серцево-судинними захворюваннями, ІХС, алгоритмами та методами машинного навчання, інтелектуальним аналізом даних і ШІ. Для пошуку використовувалась платформа Semantic Scholar, із застосуванням булевих операторів (AND, OR тощо). З понад 600 знайдених публікацій були відібрані 40 публікацій відповідно до встановлених критеріїв.

Для оцінки точності моделей у сфері машинного навчання важливим є формування якісних наборів даних. У цьому контексті було проаналізовано низку відкритих наборів, зокрема Cleveland [13, 14], Framingham Heart Study

[15, 16], Z-Alizadeh Sani, South African Heart Disease, Statlog Disease, Long Beach і KNHANES. Вони доступні в репозиторії UCI Machine Learning Repository і широко застосовуються у численних дослідженнях. Розглянемо ці набори даних.

1. Cleveland (UCI) – найчастіше використовуваний набір даних, що згадується у багатьох дослідженнях [17–20]. Він включає інформацію про 303 пацієнтів із серцевими захворюваннями. З 76 ознак більшість дослідників обмежуються 14 основними, серед яких: стать, вік, максимальна частота серцевих скорочень, тип болю в грудях, цукру в крові натще, рівень холестерину, артеріальний тиск у спокої, результати ЕКГ, стенокардія при фізичному навантаженні, депресія сегмента ST, кількість уражених судин, діагноз серцевого захворювання.

2. Framingham Heart Study [15, 16] – це набір, що доступний на платформі Kaggle, включає понад 4000 записів інформації з кардіологічного дослідження мешканців Массачусетсу і Фремінгему і використовується для аналізу 10-річного ризику розвитку ІХС. Він містить 15 ключових індикаторів, зокрема: поведінкові, медичні та демографічні фактори ризику.

3. Z-Alizadeh Sani (UCI) – дуже популярний набір, що згадується в роботах [19, 21, 22]. Він містить чотири категорії ознак: демографічні дані, симптоми та результати обстежень, ЕКГ, лабораторні та ехографічні показники. У цьому наборі пацієнтів класифікують як хворих на ІХС, якщо звуження судин становить 50% або більше.

4. Statlog Disease (UCI) – включає 13 атрибутів, таких як: вік, стать, тип болю в грудях, артеріальний тиск у стані спокою, рівень холестерину, максимальна частота серцевих скорочень, рівень цукру в крові (натще > 120 мг/дл), результати ЕКГ, депресія сегмента ST, кількість судин, виявлених флюорографією, тип таласемії.

5. KNHANES-VI – містить понад 25000 записів, отриманих із Корейського національного дослідження здоров'я та харчування. Його використовували для побудови прогнозних моделей ІХС.

6. У дослідженні [19] запропоновано об'єднати два окремих набори даних для підвищення точності алгоритмів машинного навчання у діагностиці ішемічної хвороби серця (ІХС). У цьому підході один набір даних використовувався для навчання моделі, тоді як інший слугував тестовим набором. Такий підхід дозволив застосувати метод перехресної класифікації, яка забезпечила перевірку правил, отриманих на одному наборі, на іншому незалежному. Це підвищило достовірність та надійність результатів.

Ці набори даних забезпечують необхідну основу для тестування алгоритмів машинного навчання та підтримки прогресу у вдосконаленні методів діагностики ішемічної хвороби серця.

Проаналізовано алгоритми діагностики ІХС, що обираються дослідниками, результати яких оцінювалися за показниками точності, чутливості, специфічності та F1 міри. Серед застосованих наборів даних – Z-Alizadeh Sani, Cleveland, Statlog, Framingham Heart Study тощо.

Аналіз показав, що найпоширенішим алгоритмом є метод опорних векторів (SVM), який застосовується у 70% розглянутих досліджень. Інші популярні алгоритми:

- дерева рішень – 68%;
- наївний байєсів класифікатор – 65%;
- ансамблеві методи (наприклад, XGBoost) – 55%;
- метод k-найближчих сусідів – 50%;
- алгоритм випадкового лісу – 50%.

Моделі, такі як логістична регресія (LR), нейронні мережі та інші ансамблеві методи, демонструють високу ефективність, але використовуються менш часто.

В проаналізованих дослідженнях автори використовували наступний інструментарій:

- Python (з бібліотеками Scikit-learn, TensorFlow, Keras);
- R (включаючи RStudio);
- MATLAB, WEKA, Rapid Miner.

Загальний висновок свідчить про те, що алгоритми машинного навчання здатні досягати високої точності у діагностиці ІХС, що робить їх перспективними для застосування в практичній медицині. Використання цих алгоритмів сприяє зниженню ризиків, точнішій діагностиці та ефективнішому лікуванню пацієнтів із серцевими захворюваннями.

Отже, в якості вихідних даних власного дослідження прийнято рішення використовувати набір даних Framingham Heart Study, який отримано з активного кардіологічного дослідження мешканців Массачусетсу та Фремінгему, а також комбінацію датасетів з бази даних UCI Machine Learning Repository, створених чотирма кардіологічними центрами Угорщини, Швейцарії та США. Первинний аналіз даних показав, що дані мають наступні особливості:

- наявність пропусків у даних із певними патернами;
- значна кількість якісних порядкових ознак.

Перерахуємо етапи обробки даних та аналізу.

1. Підготовка даних:

– перетворення якісних ознак: якісні порядкові змінні (наприклад, тип болю в грудях чи рівень таласемії) буде закодовано в кількісні значення, наприклад, методом label encoding;

– імпутування пропусків.

2. Побудова моделі класифікації:

– модель випадкового лісу (Random Forest);

– підбір найкращих гіперпараметрів за рахунок оптимізації через Grid Search CV: кількість дерев у лісі (`n_estimators`), максимальна глибина дерева (`max_depth`), кількість ознак для розгляду під час розбиття (`max_features`), мінімальна кількість зразків у вузлі (`min_samples_split`);

– оцінка ключових метрик якості моделі: точність (`accuracy`), чутливість чи повнота (`recall`).

3. Аналіз впливу імпутування:

– дослідження точності класифікації та швидкодії алгоритмів:
виконати експерименти з різними техніками імпутування;

– дослідження впливу методу імпутування на точність класифікації:
оцінити їх вплив на показники моделі (точність, чутливість).

4. Очікуваний результат:

– підготовка даних та застосування методів імпутування дозволить отримати точнішу модель, яка враховує характеристики пропусків та наявність якісних ознак.

1.2 Огляд задачі прогнозування рівня та витрат води басейну ріки Дніпро

Вихідними даними цієї частини дослідження є багаторічні спостереження про режим та ресурси поверхневих вод суші басейну Дніпра [23]. Використано датасет гідрометеорологічної служби України, зібраний у період з 1991 по 2014 рік, наданий НДІ геології Дніпровського національного університету імені Олеся Гончара та науковою школою проф. Шерстюк Н.П. [24, 25]. Мережа спостережень охоплює 143 пости (рисунок 1.1).

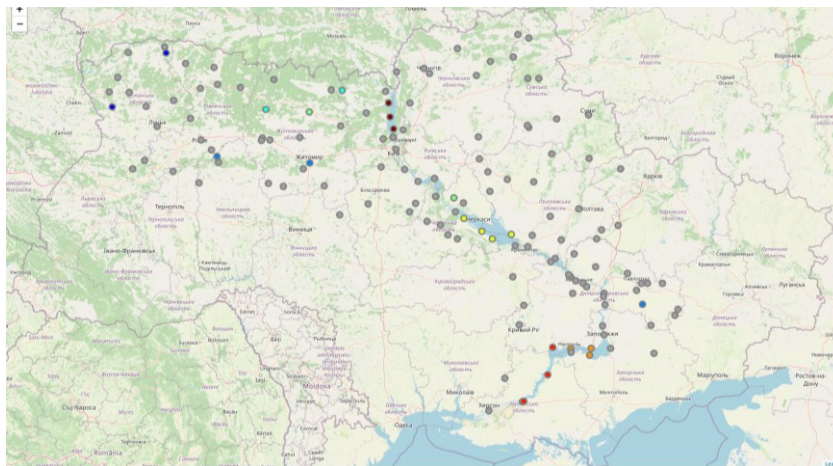


Рисунок 1.1 – Мережа спостережень за гідрологічними показниками басейну ріки Дніпро

Неструктурований набір даних, представлений таблицями Excel та файлами Word, перетворено Батурінець А.Г. у реляційну базу даних

hydro_monitoring [26], що дозволяє зручно аналізувати часові ряди та виконувати прогнозування.

Спостереження виконувались кожного дня, відповідно ми маємо справу з часовими рядами. Типовою задачею є задача прогнозування даних спостережень на наступний найближчий період. Дані мають сезонну структуру. І цей факт потребує обов'язково повних даних, для того, щоб вірно визначити період і враховувати його в моделях прогнозування.

Первинний аналіз даних показав наявність пропусків, тому майбутній аналіз буде направлений на отримання можливості заповнити ці пропуски. Дане дослідження присвячено пошуку можливих підходів до імпутування пропусків у даних в рамках заданих умов.

В роботах [27, 28] пропонується методика подовження таких рядів шляхом пошуку схожих гідрологічних рядів за допомогою метрик відстаней (евклідова, манхеттенська) і кореляційного аналізу [29, 30]. В них використовується інформація про наявність просторових залежностей. Проте дана робота буде виходити з наявної сезонної залежності та кореляційних зв'язків між показниками.

Первинному аналізу даних та пошуку ідей щодо заповнення пропусків у даних присвячено наші роботи [105, 110–113], результати яких було представлено на Всеукраїнському конкурсі студентських наукових робіт у 2020/2021 навчальному році за спеціальністю «Інформаційні системи і технології» (переможець II туру конкурсу, диплом III ступеня, м. Хмельницький, 2021 рік).

Для проведення статистичного аналізу даних нам потрібно перейти від часових рядів до випадкових величин. У цьому контексті в якості випадкових величин розглядаються рівень води за конкретний місяць року та витрати води за той самий місяць:

1. Виділяються дані за рівнем і витратами води для кожного окремого поста спостережень та місяця за всі роки.

2. Формується окремий набір для аналізу (наприклад, усі січневі дані за 1991–2014 роки конкретного посту спостережень).

При первинному аналізі даних виявлено пропуски у даних. Отже, наступні етапи аналізу будуть направлені на пошук способу заповнення цих пропусків:

1. Провести статистичний аналіз даних для вибірок, що відповідають кожному місяцю року.

2. Пошук груп вибірок з постів спостережень, в межах яких можна виконати спільне імпутування пропусків:

– знайти групи вибірок, для яких збігаються емпіричні функції розподілу;

– знайти групи однорідних вибірок.

3. Провести кореляційний і регресійний аналіз з метою виявлення залежностей між показниками, які можна було б врахувати для більш точного імпутування пропусків.

4. Запропонувати можливі способи заповнення пропусків даних на основі отриманих результатів.

Було сформовано сім груп постів, для яких емпіричні розподіли збігаються за критерієм Смірнова-Колмогорова. Деякі з цих груп включають пости, що географічно розташовані на значній відстані один від одного, що вимагає додаткового аналізу за участі фахівців-гідрогеологів. Якщо аналіз підтвердить однорідність таких груп, то дані з постів усередині кожної групи можна буде об'єднувати або використовувати для заповнення пропусків у даних.

Під час перевірки однорідності вибірок необхідно встановити, чи відповідають їх розподіли нормальному. Якщо нормальний розподіл підтверджується, можна застосовувати параметричні критерії однорідності, які аналізують збіг середніх значень та дисперсій. У разі, якщо нормальний розподіл не підтверджується, доцільно використовувати непараметричні

методи, такі як рангові критерії Манна-Уїтні або Вілкоксона, що є аналогами параметричних критеріїв.

Для визначення постів із нормальним розподілом аналіз проводився в декілька етапів. Спочатку було перевірено вірогідність розподілів, представлених у бібліотеці Pandas, використовуючи критерій згоди Смірнова-Колмогорова. Далі застосовувались додаткові тести, такі як Шапіро-Вілка, Д'Агостіно та Андерсона-Дарлінга, для підтвердження нормального розподілу. Якщо за одним із критеріїв розподіл вибірки відповідав нормальному, пост додавався до списку постів із нормальним розподілом.

Критерії однорідності перевіряють гіпотезу про збіг функцій розподілу. Для постів зі списку з нормальним розподілом використовуються параметричні методи, що аналізують середні значення та дисперсії. Для інших постів застосовують непараметричні критерії, такі як Манна-Уїтні та Вілкоксона. За результатами перевірки гіпотези однорідності формуються нові групи об'єктів зі збігом функцій розподілу.

Для рівня води у січні визначено пости спостережень, що підтверджують нормальний розподіл: 80802, 80805, 79518, 79521, 80127, 80131, 80259, 80298, 80391, 80987, 80965, 79361, 79365, 79403, 79584, 80133, 80436, 80909. Для цих постів застосовано параметричні критерії однорідності, а для інших — непараметричні методи. На рисунку 1.2 представлені групи однорідних вибірок за критерієм Манна-Уїтні, з відображенням груп різними кольорами, а пости спостережень, що не входять до жодної групи, позначені сірим кольором.

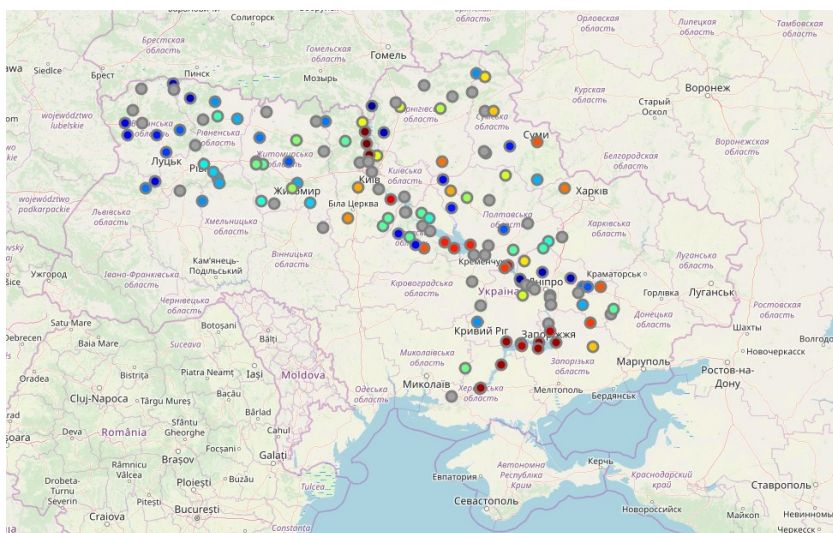


Рисунок 1.2 – Мапа груп однорідних вибірок згідно критерію Манна-Уїтні

Переважно пости в межах однієї групи розташовані поруч, однак іноді вони знаходяться далеко один від одного. У таких випадках необхідно проконсультуватися з фахівцями для підтвердження можливості використання цих груп у розрахунках. Якщо даних у якомусь посту спостережень бракує, їх можна заповнити даними з іншого поста тієї ж групи. Так пропущене значення можна замінити усередненим значенням за іншими постами групи. Такий підхід аналогічний ручному методу заповнення пропусків у даних гідрологічного моніторингу за допомогою каталогів рік-аналогів. Враховуючи зміни в кліматі та географії, ці каталоги, створені у середині ХХ сторіччя, менш актуальні в умовах сучасної цифровізації.

Однак, аналіз показав, що знайдено не досить багато груп для використання цієї ідеї імпутування пропусків. Тому для дослідження іншого підходу проаналізуємо залежності між показниками про рівень води та витрати води. Для цього проведемо кореляційний та регресійний аналіз за наступним планом:

1. Побудуємо кореляційні поля для візуальної оцінки зв'язку.
2. Обчислимо коефіцієнти кореляції Пірсона, Спірмена, Кендала для виявлення лінійного або монотонного зв'язку.

3. Побудуємо регресійні залежності згідно виявленому типу.

Для кожного поста було виконано кореляційний і регресійний аналіз, що виявив наявність статистично значущих кореляцій між показниками. На основі цього було побудовано різноманітні моделі регресії (рисунок 1.3).

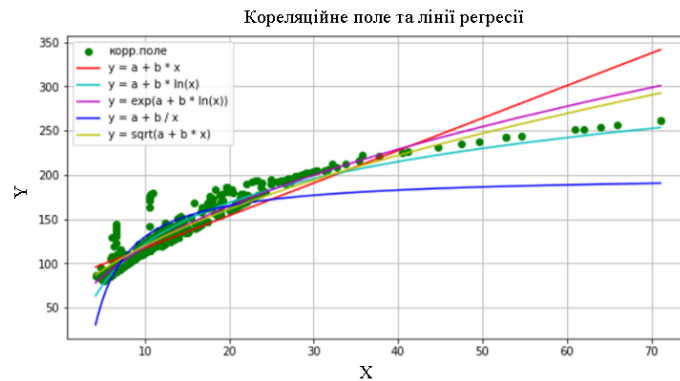


Рисунок 1.3 – Результати порівняльного аналізу моделей регресії

Серед них виділено кілька моделей, які демонструють гарну точність, і вони дозволяють прогнозувати значення одного показника, виходячи зі значень іншого. Таким чином, ці моделі регресії стануть основою для розробки алгоритму імпутування. Такий імпутор дозволить заповнювати пропущені дані з урахуванням виявлених закономірностей.

1.3 Огляд методів імпутування пропусків у даних

Пропущені значення зазвичай пояснюються людською помилкою при обробці даних, машинною помилкою через несправність обладнання, відмовою респондентів відповідати на певні запитання, вибуттям з дослідження та об'єднанням непов'язаних даних [31, 32]. Проблема пропущених значень, як правило, поширена в усіх сферах, що мають справу з даними, і спричиняє різні проблеми, такі як погіршення продуктивності, проблеми з аналізом даних та упереджені результати, спричинені різницею у пропущених та повних значеннях [33]. Більше того, серйозність пропущених значень частково залежить від того, скільки даних не вистачає, закономірності пропущених даних та механізму, що лежить в основі пропущених даних [34].

Пропущені значення можна усунути за допомогою певних методів, включаючи видалення прикладів і заміну їх потенційними або оціночними значеннями [35–37].

Заміна пропущених даних на оціночні називається *імпутацією пропусків* у даних [38].

Існує декілька традиційних статистичних методів та методів машинного навчання, таких як середнє значення, регресія, k -найближчих сусідів, ансамблеві тощо, для обробки відсутніх значень [39, 40]. У деяких випадках також використовувалися гібридні підходи [41–45] для усунення недоліків традиційних методів імпутації. Однак, важливо зазначити, що єдине прийнятне рішення зводиться до хорошого дизайну та якісного аналізу [46]. Це пов'язано з тим, що аналіз ефективності залежить від декількох факторів, таких як тип обраного алгоритму, вибір атрибутів та вибірки. Крім того, оскільки настала ера великих даних, дані стали великими і складними, зараз важко мати справу з відсутніми даними, використовуючи традиційні методи навчання, оскільки усталений процес навчання традиційними методами не був розроблений для роботи з великими даними [47]. Тому при роботі з пропущеними даними підхід завжди має вирішальне значення, оскільки неправильне поводження може привести до неточних висновків.

Механізми пропущених даних. Здебільшого механізми, які призводять до пропущених значень у даних, впливають на деякі припущення, що підтримують більшість методів обробки пропущених даних, отже, в літературі пропущені дані визначаються відповідно до цих механізмів.

Автори робіт [48, 49] запропонували вже загально прийняту класифікацію пропусків у даних:

1. Пропуски типу MAR (Missing At Random) – коли пропуски у даних залежать від спостережуваних значень, але не залежать від самих пропущених значень. Наприклад, пропуски можуть виникати частіше у певних групах, але вони не залежать від конкретних пропущених значень. Імпутування за допомогою спостережуваних змінних може бути ефективним.

2. Пропуски типу MCAR (Missing Completely At Random) – коли пропуски є випадковими і не залежать ні від спостережуваних, ні від пропущених значень. Це найбільш бажана ситуація.

3. Пропуски типу MNAR (Missing Not At Random) – коли пропуски залежать від самих пропущених значень. Наприклад, якщо люди з високим рівнем доходу частіше не вказують свій дохід, пропуски залежать саме від цього значення. Імпутування таких даних є найбільш складним.

На думку авторів робіт [39, 50], здебільшого неможливо однозначно класифікувати пропущені дані за цими трьома механізмами, оскільки уявити, що пропущені дані повністю не пов'язані з іншими не пропущеними, дуже складно. Так чи інакше пропущені значення пов'язані з не пропущеними.

Підходи з вирішення проблеми відсутніх значень. В роботі [51] запропоновано три типи технік імпутування: видалення пропусків, статистичні техніки та техніки на основі машинного навчання. Видалення пропусків – це найпростіший метод, коли пропуски видаляють та розглядають лише повний набір даних. Статистичні техніки включають заповнення середнім, модою чи найчастішим значенням (Simple imputer [89]), метод максимізації очікувань (EM-алгоритм), методи хот-дек (Hot deck), байєсовські методи, методи множинної імпутації [49, 52], компонентний аналіз [49, 53]. Звичайно при аналізі методів імпутування рішення за датасетами з видаленими даними та заповненими Simple imputer виступають як базові моделі для порівняння.

Методи на основі машинного навчання охоплюють штучні нейронні мережі [54, 55], асоціативні правила, кластеризацію [56, 57], дерева рішень [58], випадковий ліс [59], машину опорних векторів [60] та регресійний підхід [61, 62].

Видалення. У цьому підході всі записи з пропущеними значеннями видаляються/відкидаються при проведенні аналізу. Видалення вважається найпростішим підходом, оскільки немає необхідності намагатися оцінити значення. Однак автори роботи [49] продемонстрували деякі слабкі сторони видалення, оскільки воно вносить упередженість в аналіз, особливо коли пропущені дані не є

випадково розподіленими. Процес видалення може здійснюватися двома способами: попарне видалення або видалення за списком [63].

Видалення за списком. При видаленні за списком видаляється кожен приклад (об'єкт), який має одне або декілька відсутніх значень. Видалення за списком стало вибором за замовчуванням при аналізі даних у більшості статистичних програмних пакетів [64]. Однак, якщо припустити, що дані не є MCAR, то видалення за списком призводить до зсуву [65]. Якщо ж вибірки даних достатньо великі і припущення MCAR виконується, то видалення за списком може бути розумним підходом. Якщо вибірка даних невелика або припущення MCAR не виконується, то видалення за списком не є найкращим підходом. Видалення за списком також може привести до втрати важливої інформації, особливо якщо відкинуті приклади мають велику кількість.

Попарне видалення. Щоб запобігти втраті інформації під час вилучення за списком, можна скористатися попарним вилученням. Попарне видалення виконується таким чином, щоб зменшити втрати, які можуть виникнути під час видалення за списком. Процедура не може включати певну змінну, якщо вона має пропущене значення, але вона все одно може використовувати цей випадок під час аналізу інших змінних з непропущеними значеннями. Датасет може містити 3 ознаки: x_1 , x_2 і x_3 . Об'єкт може мати відсутнє значення для x_1 , але це не заважає деяким статистичним процедурам використовувати цей об'єкт для аналізу змінних x_2 і x_3 . Попарне видалення дозволяє використовувати більше даних. Однак кожна обчислена статистика може базуватися на різній підмножині прикладів. Це може виявитися проблемою [66–68].

Імпутація. Процес імпутації передбачає заміну відсутніх значень деякими прогнозованими значеннями. Набір даних без відсутніх значень зазвичай використовується для прогнозування значень, які використовуються для заміни відсутніх значень [38]. Далі ми розглядаємо деякі з найбільш використовуваних методів імпутації в літературі.

Проста імпутація (Simple imputation). Простий підхід імпутації передбачає заміну відсутніх значень для кожного окремого значення за допомогою кількісного атрибута або якісного атрибута всіх непропущених значень [69]. За допомогою простого імпутування відсутні дані замінюються різними методами, такими як мода, середнє або медіана доступних значень. У більшості досліджень використовуються прості методи імпутації через їхню простоту та те, що їх можна використовувати як простий довідковий метод [70]. Однак прості методи імпутації можуть призвести до зсуву або нереалістичних результатів на наборах даних великої розмірності [71].

Hot-deck imputation. Метод хот-дек (hot deck) має низку модифікацій [72–75]. Найпростіший варіант – сортування респондентів за ключовими змінними, тоді респонденти зі схожими відповідями знаходяться поруч один з одним. Як ключові найчастіше виступають соціально-демографічні змінні, але можна використовувати й інші змінні, що мають кореляції зі змінною з пропущеними даними. При відновленні пропущені значення змінної запозичуються із попереднього спостереження. В окремих модифікаціях хот-дек респонденти сортуються для кожної ознаки за допомогою різних наборів змінних для попередження взаємозалежностей між значеннями, що відновлюються. Найбільш розвиненою модифікацією вважається хот-дек з випадковим відбором значень з підгрупи подібних респондентів.

Респонденти поділяються на підгрупи за ключовими змінними. У середині кожної підгрупи дані заміни пропущених відповідей відбираються випадковим чином, але з тим самим розподілом, як і відповіді у підгрупі. Головним недоліком методу хот-дек є наявність взаємозв'язків між відновленими значеннями та зниження значень дисперсії. Зменшити рівень залежності до незначних величин можна лише за наявності великої кількості підгруп, що передбачає великі обсяги вибірок.

У роботі [74] запропоновано метод хот-дек, який дозволяє дослідити вплив механізмів пропусків, починаючи від MAR і закінчуючи MNAR, і використовує інформацію, що міститься в повністю спостережуваних даних.

Зсув оцінок було досліджено шляхом моделювання. Результати також показали, що метод працював найкраще, коли повністю спостережувані значення були пов'язані з результатом.

Expectation-maximization (EM) – це ітераційний метод для обробки відсутніх значень у числових наборах даних, який використовує підхід «імпутація, оцінка та ітерація до збіжності». Кожна ітерація складається з двох етапів: очікування (E) та максимізація (M). Очікування оцінює відсутні значення за даними спостережень, тоді як при максимізації наявні оцінені значення використовуються для максимізації ймовірності всіх даних [76].

У роботі Рубіна та ін. [77] проведено дослідження щодо обробки відсутніх даних з використанням набору даних, який аналізував вплив харчової поведінки серед тварин, які отримували медикаментозне лікування та нелікованих тварин. Результати імпутування за EM-алгоритмом порівнювалися з іншими методами, такими як видалення за списком, який виявився найменш ефективним, байєсівським підходом та регресією. Автори дійшли висновку, що алгоритм EM є найкращим методом для типу даних, які вони використовували. Однак використання реальних наборів даних у дослідженні могло призвести до того, що результати були специфічними для особливостей набору даних та вибірки або відображають гіпотетичні очікування.

В іншому дослідженні EM-алгоритм використовувався у даних для вирішення проблеми навчання гаусових сумішей у великих наборах даних високої розмірності з пропущеними значеннями [78]. Після імпутування отримані дані були використані в моделях класифікації, і виявилось, що вони забезпечують значне покращення продуктивності порівняно з іншими базовими методами імплікації пропущених значень.

Множинне імпутування. Очевидно, що обробка пропущених даних виходить за рамки видалення або відкидання пропущених даних [48], і тому дослідники вдаються до множинної імпутації. Ідея множинного імпутування виходить з того, що розподіл спостережуваних даних використовується для

апроксимації числових значень, які відображають невизначеність навколо істинного значення, і цей метод здебільшого застосовувався для подолання обмежень одиночної імпутації [79]. Аналіз проводиться на наборі даних з використанням різних методів обробки пропущених даних, а середнє значення оцінок параметрів для m вибірок буде давати нам одну точкову оцінку. Таким чином, метод множинної імпутації складається з трьох окремих етапів:

- відсутні дані обробляються в m вибірках, в результаті чого отримується m повних наборів даних;
- потім аналізуються m повних наборів даних;
- результати всіх m наборів імпутованих даних об'єднуються для отримання остаточного результату імплікації.

Хоча множинне імпутування є стандартною методологією для заповнення пропущених значень, важливо, щоб дослідники використовували відповідні методи для імпутації, щоб гарантувати отримання надійних результатів під час експериментів з цим підходом [80]. На результативність може негативно вплинути проведення імпутації на реальних даних, таких як дані опитувань, клінічні дані та промислові дані, які можуть характеризуватися високим рівнем пропусків і великою кількістю факторів, які не обов'язково лінійно пов'язані між собою. Крім того, традиційні методи множинної імпутації погано працюють з даними високої розмірності, і дослідники вдаються до вдосконалення цих алгоритмів, щоб підвищити їх ефективність [81, 82]. Існують також докази того, що слід з обережністю застосовувати методи на основі неперервних даних при інтерполяції категоріальних даних, оскільки це може призвести до упереджених результатів [83].

Дослідники в роботі [83] експериментували з методом, який точно імпутував відсутні значення в наборі даних про пацієнтів за допомогою множинної імпутації з використанням методу найменших квадратів опорного вектору (LSSVM). Для визначення ефективності запропонованого методу було використано п'ять наборів даних. Результати оцінки показали, що їхній метод перевершує традиційні методи імпутації, і що дослідження було більш

надійною методикою, яка генерує значення, ближчі до того, якого не вистачало. Крім того, автор також запропонував інший метод – Clustered Z-score Least Square Support Vector Machine (CZLSSVM) і продемонстрував його ефективність у двох задачах класифікації для неповних даних. Їхні експериментальні результати також показали, що точність класифікації збільшилася за допомогою CZLSSVM, і що алгоритм перевершив інші підходи до інтерполяції даних такі як SVM, дерево рішень, KNN, грубі множини та штучні нейронні мережі. В іншому дослідженні [84] автори запропонували метод множинної імплікації для даних клінічної практики. Результати методу показали гарні результати для пропусків за механізмами MCAR або MAR. В роботі [85] автори досліджували метод множинної імпутації, який розширював багатовимірну імпутацію за допомогою ланцюгового рівняння для великих даних. Підхід був представлений у двох варіантах, один для категоріальних, а інший – для числових даних і реалізував дванадцять існуючих алгоритмів для порівняння продуктивності. Експериментальні результати з чотирма наборами даних продемонстрували, що метод працює найкраще при інтерполяції бінарних та числових даних.

У цьому огляді представлено комплексний огляд проблеми відсутніх значень, включаючи механізми відсутніх даних та підходи до вирішення цієї проблеми для різних сценаріїв. Це дослідження привело до висновку, що точність алгоритмів імпутації сильно залежить від типу даних, що аналізуються, і що немає чітких ознак переваги одного методу над іншим. Огляд продемонстрував існування обмежень на існуючі методи відсутніх значень. Крім того, більшість розглянутих робіт аналізують різні невеликі набори даних. Натомість датасети з реального життя містять велику кількість різноманітних ознак. Тому потрібна подальша розробка нових методів обробки відсутніх даних з урахуванням можливих зв'язків між ознаками, патернів у даних та інших особливостей.

1.4 Висновок до розділу 1

У науці про дані дослідницький аналіз відіграє ключову роль на початковому етапі роботи. Він спрямований на вивчення властивостей даних, виявлення закономірностей та аномалій, очищення від викидів і створення базових моделей для подальшого аналізу. Під час цього етапу визначають тип розподілу даних, оцінюють його основні параметри, знаходять викиди, створюють кореляційну матрицю між ознаками тощо.

Однією з головних проблем у первинному аналізі є наявність пропущених значень. Найбільша складність полягає в тому, що для заповнення пропусків не існує універсального підходу. Для кожної задачі потрібно підбирати відповідні методи, поєднувати їх, адаптувати існуючі або навіть розробляти нові способи вирішення.

Кожен набір даних має свої унікальні виклики, тому важливо підбирати відповідні методи обробки залежно від специфіки задачі. У цьому контексті розглянемо дві типові задачі інтелектуального аналізу даних: класифікацію та прогнозування часових рядів.

Як приклад задачі класифікації розглядається діагностика ішемічної хвороби серця, тоді як прогнозування часових рядів буде виконуватись на основі даних гідрологічного моніторингу. Саме під призмою цих задач будемо розглядати проблеми у даних і задачі, пов'язані з перетворенням якісних ознак на кількісні та імпутованням пропущених даних.

РОЗДІЛ 2. АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ

В ході дослідження розроблено декілька методів та алгоритмів імпутування пропусків у даних. Для перетворення категоріальних ознак на кількісні зі збереженням інформації про пропуски даних розроблено два методи `IgnoreNaNLabelEncoder` та `IgnoreNaNFrequentEncoder`, представлені в підрозділі 2.1. Підрозділ 2.2 містить опис метода імпутування `UnifiedClassRegrImputer` на основі застосування класифікатору та регресору. Підрозділ 2.3 присвячений методу `RegrImputer` на основі застосування регресору. В розділі 2.4 наведений метод імпутування `EntropyImputer` на основі ентропійного підходу. В розділі 2.5 представлено гібридний метод імпутування `HybridRegrEntropyImputer`, що поєднує ентропійний та регресійний підходи. В підрозділі 2.6 наведено метод імпутування `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками. Висновки до розділу наведено в підрозділі 2.7.

2.1 Методи перетворення якісних ознак на кількісні

Існує кілька основних методів [86] для перетворення якісних (категоріальних) ознак на кількісні (числові) для подальшого використання в алгоритмах машинного навчання:

1. `Label Encoding`: Кожна категорія перетворюється на унікальне ціле число. Цей метод застосовують для порядкових категорій (ознак, що мають природний порядок, наприклад, освітній рівень). Однак для номінальних ознак, де порядок не має значення, цей метод може призвести до неправильних результатів, оскільки моделі можуть сприймати числові значення як певну ієрархію.

2. `One-Hot Encoding`: Кожна категорія перетворюється на окремий бінарний стовпець (0 або 1). Використовується для номінальних ознак, де немає порядку між категоріями. Недоліком цього методу є збільшення

кількості стовпців (особливо при великій кількості категорій), що може призвести до надмірного споживання пам'яті.

3. Ordinal Encoding: Кожна категорія перетворюється на ціле число відповідно до визначеного порядку. Підходить для ознак з чіткою ієрархією, наприклад, рівень задоволеності або ризику. Важливо, щоб порядок чисел відповідав природному порядку категорій.

4. Binary Encoding: Поєднання Label Encoding та бінарного представлення чисел. Кожне числове значення категорії кодується в бінарному вигляді, і кожен біт представлений окремою колонкою. Binary Encoding дозволяє зменшити кількість стовпців порівняно з One-Hot Encoding і є ефективним при роботі з ознаками з багатьма категоріями.

5. Target Encoding (Mean Encoding): Кожна категорія замінюється середнім значенням цільової змінної для цієї категорії. Підходить для задач прогнозування, однак є ризик перенавчання (особливо при невеликій кількості категорій). Застосовується переважно у задачах класифікації.

6. Frequency Encoding: Кожна категорія замінюється частотою її появи в наборі даних. Це особливо корисно, коли потрібно врахувати частотність категорії, однак не завжди підходить для ознак з рівнозначними категоріями.

Кожен з методів має свої переваги і недоліки, і вибір залежить від типу категоріальної ознаки (номінальна чи порядкова), обсягу даних та вимог до моделі.

Найбільш поширеними методами для кодування якісних ознак є Label Encoding та One-Hot Encoding. Розглянемо Label Encoding: цей метод перетворює унікальні значення ознак у числа $0, 1, 2, \dots, k - 1$, де k – кількість різних значень ознаки. У Python цей метод реалізовано за допомогою класу `LabelEncoder` з бібліотеки `scikit-learn` (модуль `sklearn.preprocessing`) [87]. Проте `LabelEncoder` не передбачає наявності пропусків у даних, що ускладнює автоматизацію обробки даних, коли імпутація пропущених значень відбувається вже після перетворення ознак. Крім того, цей метод не враховує частотність кожного значення.

Модифіковано стандартний метод `LabelEncoder` з бібліотеки `scikit-learn` та отримано два удосконалені методи `IgnoreNaNLabelEncoder` та `IgnoreNaNFrequentEncoder` [107] перетворення якісних ознак на кількісні:

- `IgnoreNaNLabelEncoder` – модифікація стандартного `LabelEncoder`, яка не кодує пропуски, зберігає словник та дозволяє зворотне перетворення (лістинг 2.1);

- `IgnoreNaNFrequentEncoder` – модифікація `LabelEncoder` та `FrequencyEncoder`, яка враховує частоту значень, надаючи найбільший номер значенню з найбільшою або найменшою частотою в залежності від параметра `ascending`, не кодує пропуски, зберігає словник та дозволяє зворотне перетворення (лістинг 2.2).

Лістинг 2.1 – Алгоритм `IgnoreNaNLabelEncoder`

```
def label_encoding_non_missing_values(data):
    df = data.copy()
    encoders = dict()
    for col_name in df.select_dtypes(
        include=['object', 'category']).columns:
        series = df[col_name]
        label_encoder = LabelEncoder()
        df[col_name] = pd.Series(
            label_encoder.fit_transform(
                series[series.notnull()]),
            index=series[series.notnull()].index
        )
        encoders[col_name] = label_encoder
    return df, encoders

def label_decoding_non_missing_values(data, encoders):
    df = data.copy()
    for col_name in encoders:
        enc = encoders[col_name]
        nmap = dict(zip(enc.transform(enc.classes_),
            enc.classes_))
        df[col_name] = df[col_name]
            .apply(lambda x: x if np.isnan(x) else nmap[x])
    return df
```

Лістинг 2.2 – Алгоритм `IgnoreNaNFrequentEncoder`

```
def label_encoding_non_missing_values_frequent(data,
        ascending=False):
```

```

df = data.copy()
encoders = dict()
for col_name in df.select_dtypes(
    include=['object', 'category']).columns:
    d = data[col_name].value_counts().sort_values(
        ascending=ascending).to_dict()
    k = 0
    for key in d:
        d[key] = k
        k = k + 1
    encoders[col_name] = d
    df[col_name] = df[col_name]
        .apply(lambda x: x if x != x else d[x])
return df, encoders

def label_decoding_non_missing_values_frequent(data,
                                                encoders):
    df = data.copy()
    for col_name in encoders:
        d = encoders[col_name]
        nd = dict()
        for key, value in d.items():
            nd[value] = key
            df[col_name] = df[col_name]
                .apply(lambda x: x if x != x else nd[x])
    return df

```

Особливістю цих алгоритмів є створення словника для кожної ознаки, що дозволяє виконувати зворотне перетворення, а також можливість пропускати пропущені значення. Далі буде виконано рефакторинг цих алгоритмів та реалізовано у вигляді класів відповідно до архітектурних принципів бібліотеки `scikit-learn` для універсального використання. Варто зазначити, що алгоритми працюють лише зі стовпцями, тип яких належить до категорій `object` або `category`. Ці методи дозволяють одразу конвертувати всі якісні ознаки в числові без втрати інформації про пропуски.

2.2 Метод імпутовання пропусків у даних `UnifiedClassRegrImputer` та його модифікації

Розглянемо концепцію імпутації даних як задачу передбачення значень у стовпці з пропусками, використовуючи інформацію з інших стовпців, включаючи сам цільовий стовпець. Якщо цільова ознака залежить від інших ознак за певною функцією:

$$y_i = f(x_{i,1}, x_{i,2}, \dots, x_{i,k}, \dots, x_{i,p}) + \varepsilon, \quad (2.1)$$

то можна припустити, що також існує стохастична залежність k -ї ознаки від інших ознак з урахуванням цільового стовпця:

$$x_{i,k} = \varphi(x_{i,1}, x_{i,2}, \dots, x_{i,k-1}, x_{i,k+1}, \dots, x_{i,p}, y_i) + \varepsilon. \quad (2.2)$$

Цей підхід використовується у методах MICE [88, 89] та NoNa [90].

Запропоновано чотири власні реалізації цієї ідеї з деякими модифікаціями [107]. При цьому враховується, що пропуски у даних можуть мати різні патерни, а саме:

- a) відсутні рядки або стовпці, які містять повністю заповнені дані;
- b) існують n стовпців, що мають повні дані для всіх екземплярів;
- c) існують k рядків, де дані заповнені для всіх ознак;
- d) існують m рядків, де пропуск спостерігається лише в одній ознаці.

Приклад розподілу пропусків у даних наведено на рисунку 2.1.

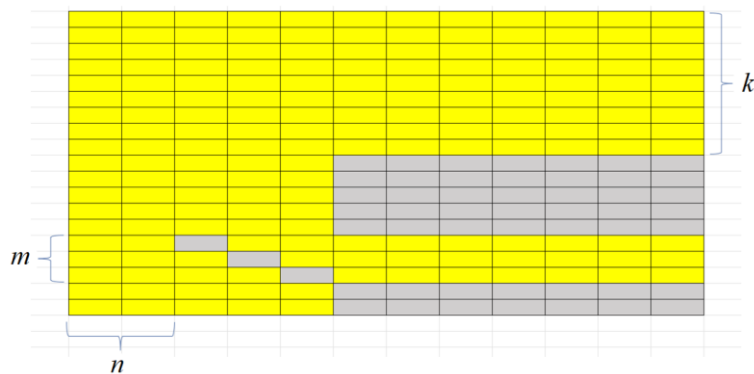


Рисунок 2.1 – Патерн пропусків у даних

(жовтий колір – наявні дані, сірий колір – пропуск у даних)

Представимо деталізацію методу у вигляді трьох алгоритмів, по одному на кожну модифікацію. Враховуємо, що ознаки можуть бути як кількісними, так і якісними (попередньо закодованими).

1. Алгоритм 2.1. Метод *fillna_k_columns* – цей підхід враховує патерни (a) і (b), застосовуючи регресор чи класифікатор залежно від типу ознаки.

2. Алгоритм 2.2. Метод *fillna_k_sorted_columns* – також враховує патерни (a) і (b), однак обробляє стовпці з урахуванням кількості пропусків у

кожному. Для цього застосовується регресор або класифікатор відповідно до типу ознаки.

3. Алгоритм 2.3. Метод *fillna_2steps_rg_class* – працює з патернами (a), (b), (c) та (d), обробляючи стовпці залежно від частоти пропусків і використовуючи регресор або класифікатор залежно від природи ознаки.

Опис алгоритмів наведено нижче.

Алгоритм 2.1 – Метод *fillna_k_columns*

1. Запускаємо цикл по стовпцях набору даних (*for j in columns*).
2. Якщо у стовпці *j* є пропущені значення, переходимо до кроку 3, інакше – до наступної ітерації (крок 1).
3. Визначаємо індекси рядків із пропусками у стовпці *j* та зберігаємо їх у масив *indexes_nan*.
4. Виділяємо підмножину *full_data*, що містить рядки без пропусків у всіх стовпцях, для прогнозування значень у стовпці *j*.
5. Якщо *full_data* порожній, заповнюємо пропуски стандартним методом: кількісні ознаки – медіаною, якісні – модою. Якщо *full_data* не порожній, переходимо до кроку 6.
6. Розділяємо *full_data* на тренувальний та тестовий набори: до тренувальної вибірки включаємо всі рядки, де у стовпці *j* є дані, тобто всі рядки, крім зазначених в *indexes_nan*.
7. Якщо стовпець *j* містить якісні дані, використовуємо класифікатор, інакше – регресор.
8. Навчаємо модель на тренувальному наборі та прогнозуємо значення для *j*-го стовпця за тестовим набором.
9. Переходимо до наступного стовпця (крок 1).

Діаграму діяльності UML для цього алгоритму наведено на рисунку 2.2.

Наступний алгоритм 2.2 (*fillna_k_sorted_columns*) є модифікацією алгоритму 2.1: він дозволяє спочатку заповнити пропуски у стовпцях із найбільшою кількістю пропусків або, навпаки, найменшою кількістю.

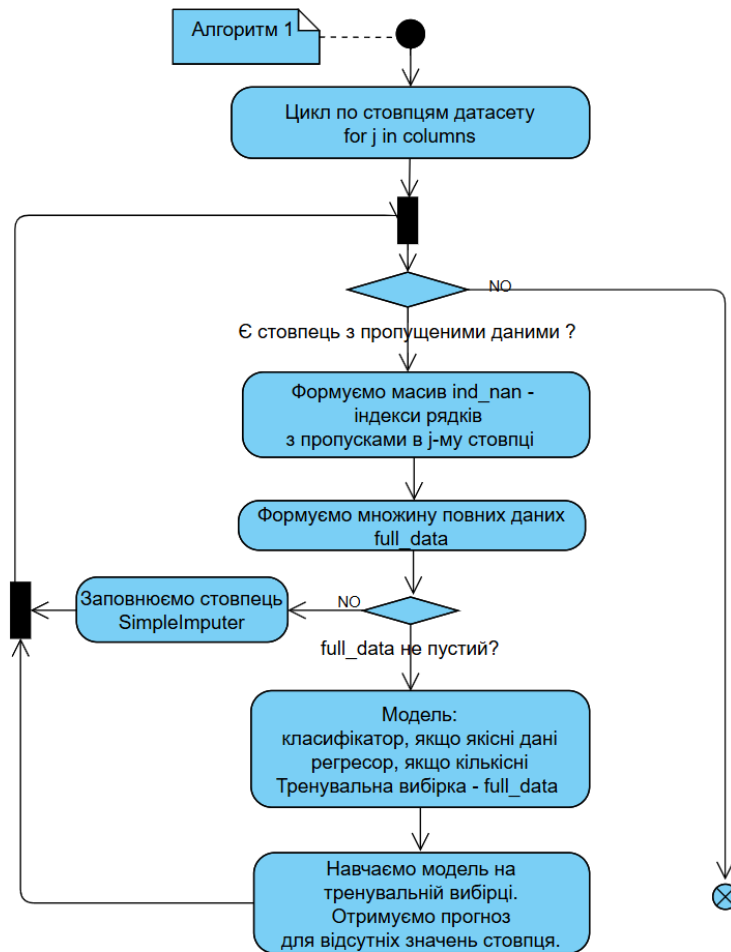


Рисунок 2.2 – Діаграма діяльності UML алгоритму 2.1 *fillna_k_columns*

Алгоритм 2.2 – Метод *fillna_k_sorted_columns*

1. Сортуємо стовпці *columns* за кількістю пропусків. Залежно від параметра *ascending* виконуємо обхід стовпців у порядку спадання або зростання кількості пропусків.
2. Запускаємо цикл по стовпцях (*for j in columns*).
3. Якщо в стовпці *j* присутні пропуски, переходимо до наступного кроку (крок 4), інакше – до наступної ітерації (крок 2).
4. Визначаємо індекси рядків, в яких присутні пропуски в стовпці *j*, і записуємо їх у масив *indexes_nan*.
5. Виділяємо підмножину *full_data*, що включає лише рядки без пропусків у всіх стовпцях. Цей датасет буде використаний для прогнозування відсутніх значень у стовпці *j*.

6. Якщо *full_data* порожній, заповнюємо пропуски стандартним методом: кількісні ознаки – медіаною, якісні – модою. Якщо *full_data* не порожній, переходимо на крок 7.

7. Розділяємо *full_data* на тренувальний та тестовий набори: тренувальний включає всі рядки, де значення для стовпця *j* не є пропущеним, тобто всі рядки, крім зазначених в *indexes_nan*.

8. Якщо стовпець *j* містить якісні дані, використовуємо класифікатор, в іншому випадку – регресор.

9. Тренуємо модель на тренувальному наборі та прогнозуємо значення для стовпця *j* на основі тестовому.

10. Повертаємося до кроку 1 для обробки наступного стовпця.

Діаграма діяльності цього алгоритму показана на рисунку 2.3, доданий блок помічено зеленим пунктиром.

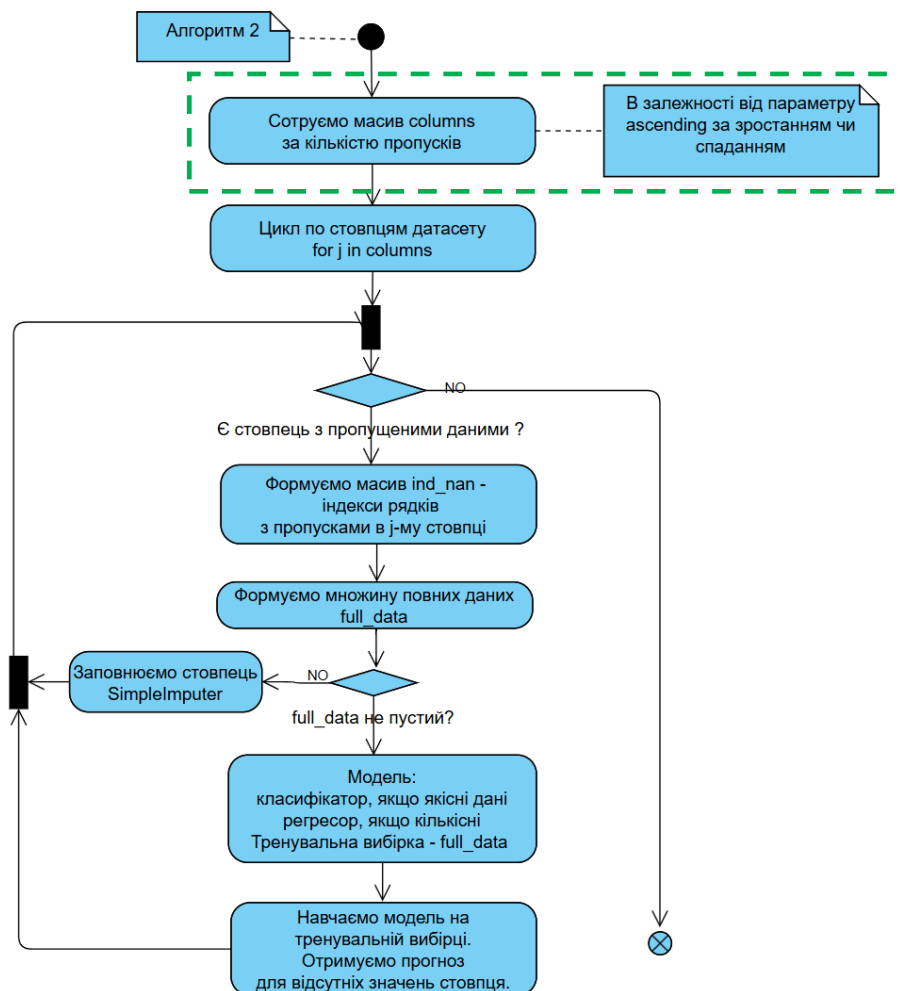


Рисунок 2.3 – Діаграма діяльності UML алгоритму 2.2 *fillna_k_sorted_columns*

Наступний алгоритм є двоетапним.

1. Перший етап: Вибираємо підмножину датасету, що складається з рядків із повністю заповненими значеннями для всіх ознак. До цієї підмножини додаємо рядки, де бракує лише одного значення. Виконуємо імпутацію даних у цих додаткових рядках за аналогією з попередніми методами, тобто враховуємо патерн (d).

2. Другий етап: Виконуємо дії, описані в алгоритмі *fillna_k_sorted_columns*, заповнюючи відсутні значення у решті рядків.

Алгоритм 2.3 – Метод *fillna_2steps_rg_class*

Етап 1.

1. Формуємо масив ознак *cols_1nan*, який містить стовпці, де у кожному рядку є лише один пропуск.

2. Якщо такий масив порожній, переходимо одразу до кроку 11.

3. Запускаємо цикл по стовпцях у *cols_1nan* (*for j in cols_1nan*).

4. Знаходимо індекси рядків із єдиним пропуском у стовпці *j* і зберігаємо їх у масив *indexes_nan*.

5. Виділяємо підмножину *full_data* датасету, що складається з рядків без пропусків, і додаємо до неї рядки з *indexes_nan*. Це стане датасетом для прогнозування пропущених значень у стовпці *j* для рядків *indexes_nan*.

6. Якщо *full_data* порожній, пропускаємо цей стовпець і переходимо до наступної ітерації циклу (крок 3).

7. Розділяємо *full_data* на тренувальну та тестову вибірки: до тренувальної включаємо всі рядки з непустими значеннями в стовпці *j*, тобто всі рядки, окрім зазначених у *indexes_nan*.

8. Якщо у стовпці *j* містяться якісні дані, використовуємо класифікатор, в іншому випадку – регресор.

9. Тренуємо модель на тренувальній вибірці і отримуємо прогнозні значення для стовпця *j* у тестовій вибірці.

10. Повертаємося до кроку 3 для обробки наступного стовпця.

Етап 2.

11. Викликаємо алгоритм *fillna_k_sorted_columns* для заповнення решти пропусків.

Діаграма діяльності цього алгоритму зображена на рисунку 2.4.

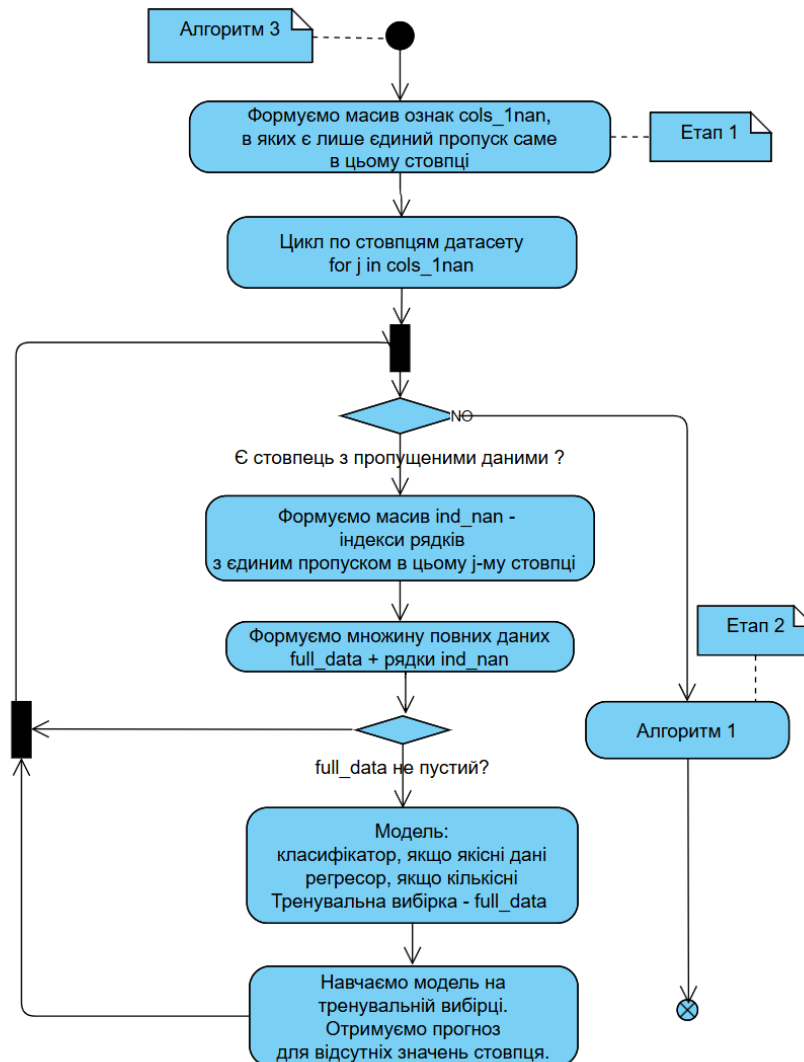


Рисунок 2.4 – Діаграма діяльності UML алгоритму 2.3 *fillna_2steps_rg_class*

Для алгоритмів *fillna_k_columns*, *fillna_k_sorted_columns* і *fillna_2steps_rg_class* проведено рефакторінг та створено уніфікований клас *UnifiedClassRegrImputer* – імп'ютер на мові програмування Python згідно архітектурним патернам *scikit-learn*.

2.3 Метод імпутування пропусків у даних `RegrImputer`

Для підвищення швидкості роботи попереднього методу спробуємо відмовитись від класифікатора. Цей метод буде застосовувати регресор для заповнення пропущених значень [107], з подальшим коригуванням у разі якісних ознак. Коригування виконується одним із двох способів: вибір найближчого значення з існуючих у словнику (параметр `measure = 'value'`) або на основі середньозваженої оцінки з урахуванням частот (параметр `measure = 'weight'`). Словник формується одразу після переходу до наступної ознаки.

Алгоритм 2.4 – Метод `fillna_2steps_rg`

Етап 1.

1. Формуємо масив ознак `cols_1nan`, що містить стовпці, де у кожному рядку є тільки один пропуск.
2. Якщо `cols_1nan` порожній, переходимо одразу до кроку 12.
3. Цикл по стовпцях у `cols_1nan` (`for j in cols_1nan`).
4. Визначаємо індекси рядків із єдиним пропуском у стовпці j і зберігаємо їх у масив `indexes_nan`.
5. Виділяємо підмножину `full_data`, що складається з рядків без пропусків, та додаємо до неї рядки з `indexes_nan`. Це буде датасет для прогнозування відсутніх значень у стовпці j для рядків `indexes_nan`.
6. Якщо `full_data` порожній, пропускаємо цей стовпець і переходимо до наступної ітерації циклу (крок 3).
7. Розділяємо `full_data` на тренувальний та тестовий набори: у тренувальному залишаємо всі рядки з непустими значеннями у стовпці j , тобто всі рядки, окрім зазначених у `indexes_nan`.
8. Незалежно від типу даних у стовпці j , використовуємо регресор.
9. Тренуємо модель на тренувальному наборі та прогнозуємо значення для стовпця j у тестовому.

10. Якщо у стовпці j містяться якісні дані, корегуємо отримані значення за допомогою одного з двох способів: *value* або *weight*.

11. Повертаємося до кроку 3 для обробки наступного стовпця.

Етап 2.

12. Сортуємо стовпці *columns* за кількістю пропусків. Залежно від параметра *ascending* обходимо стовпці у порядку зростання або спадання кількості пропусків.

13. Цикл по стовпцях (*for j in columns*).

14. Якщо у стовпці j є пропущені значення, переходимо до кроку 15, інакше – достроково до наступної ітерації (крок 13).

15. Знаходимо індекси рядків, де є пропуски в j -му стовпці, та зберігаємо до масиву *indexes_nan*.

16. Виділяємо підмножину *full_data*, де у всіх стовпцях немає пропусків, для прогнозування значень у стовпці j .

17. Якщо *full_data* порожня, заповнюємо пропуски стандартним методом: для кількісних ознак – медіаною, для якісних – модою. Якщо *full_data* не порожній, переходимо до кроку 18.

18. Розділяємо *full_data* на тренувальну і тестову вибірки: у тренувальній залишаємо всі рядки з даними у стовпці j , тобто всі рядки, крім зазначених у *indexes_nan*.

19. Використовуємо регресор у всіх випадках.

20. Навчаємо модель на тренувальній вибірці та прогнозуємо значення для стовпця j у тестовій вибірці.

21. Якщо у стовпці j містяться якісні дані, корегуємо отримані значення за допомогою одного з двох способів: *value* або *weight*.

22. Повертаємося до кроку 13 для обробки наступного стовпця.

Діаграма діяльності цього алгоритму показана на рисунку 2.5.

Для цього алгоритму виконано рефакторинг і розроблено клас *RegrImputer* згідно архітектурним патернам *scikit-learn* на Python.

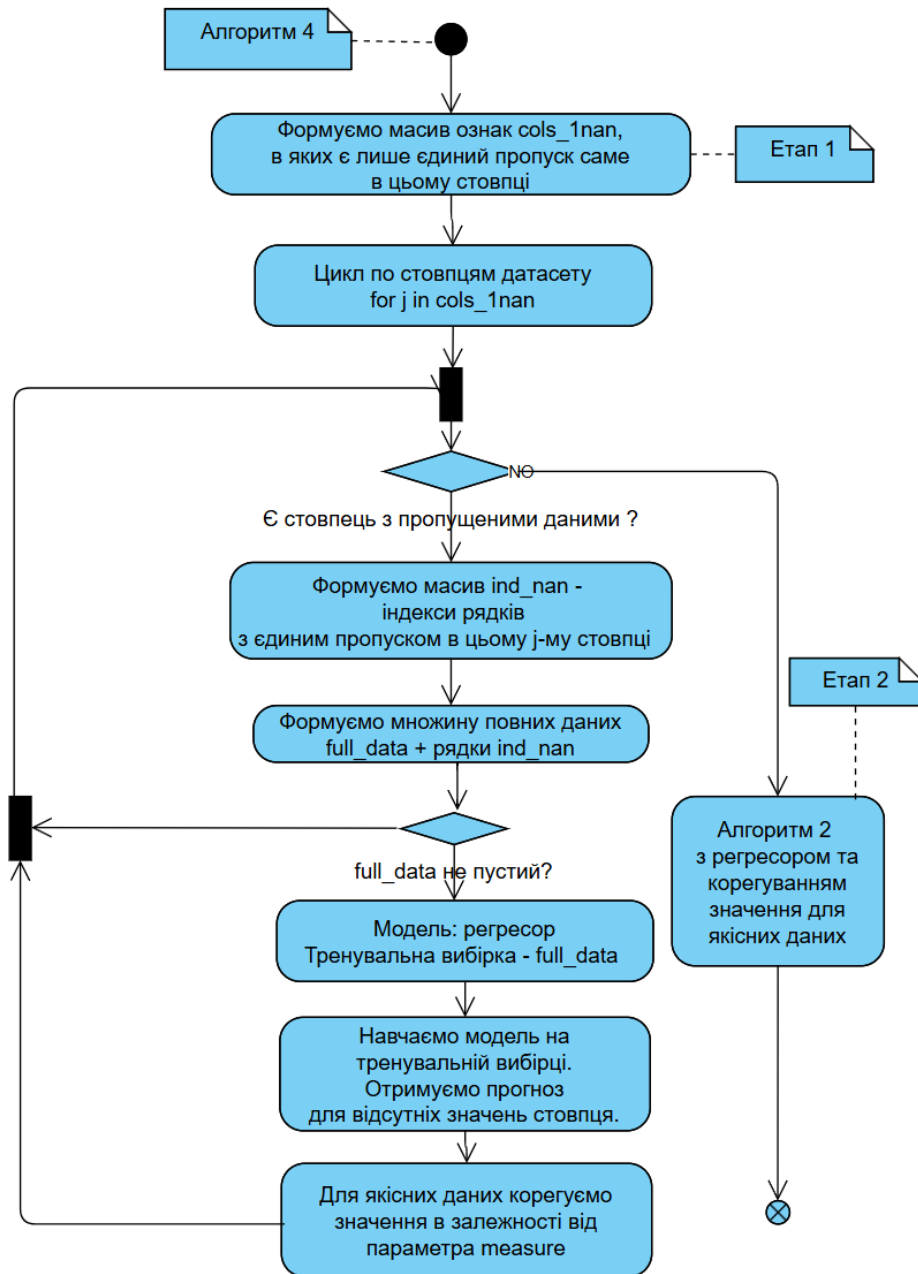


Рисунок 2.5 – Діаграма діяльності UML алгоритму 2.4 *fillna_2steps_rg*

2.4 Метод імпутування пропусків у даних *EntropyImputer*

Метод імпутування пропущених значень у даних для задач класифікації ґрунтується на ентропійному підході, що використовує поняття ентропії з теорії інформації. Ентропія характеризує рівень невизначеності або хаотичності в даних. Головна ідея методу – мінімізувати невизначеність щодо належності об'єктів до класів, обираючи такі значення для пропусків, які максимально зменшують цю невизначеність.

Згідно з теорією інформації, ентропія виступає мірою невизначеності або випадковості. Для випадкової величини Y з розподілом ймовірностей $P(Y)$ ентропія $H(Y)$ визначається як:

$$H(Y) = - \sum_{y \in Y} P(y) \log P(y). \quad (2.3)$$

У задачах класифікації ентропію можна застосовувати для оцінки невизначеності, пов'язаної з передбачуваними мітками класів на основі наявних даних. Якщо деякі ознаки відсутні, алгоритм імпутування може заповнювати пропуски так, щоб мінімізувати ентропію міток класів, що сприяє покращенню точності класифікації. Таким чином, заповнені значення допомагають зменшити невизначеність і підвищити якість передбачень моделі.

Для дискретної випадкової величини Y з можливими класами C_1, C_2, \dots, C_k та ймовірностями $P(C_1), P(C_2), \dots, P(C_k)$ ентропія $H(Y)$ визначається формулою:

$$H(Y) = - \sum_{i=1}^k P(C_i) \log_2 P(C_i). \quad (2.4)$$

Основа логарифма в формулі ентропії визначає одиницю ентропії. Як правило, використовують основу 2, і тоді ентропія вимірюється в бітах.

Для практичного розрахунку ентропії класів необхідно:

1. Визначити частоту входжень кожного класу в наборі даних.
2. Обчислити ймовірність кожного класу, поділивши кількість його входжень на загальну кількість спостережень.
3. Розрахувати ентропію розподілу класів за формулою (2.4).

У випадку набору даних з кількома ознаками ентропія класів може змінюватися залежно від значень конкретної ознаки. Для оцінки невизначеності класів за наявності певної ознаки використовують умовну ентропію.

Для ознаки X з можливими значеннями x_1, x_2, \dots, x_m умовна ентропія $H(Y|X)$ визначається як середньозважене значення ентропій за кожним можливим значенням ознаки

$$H(Y|X) = \sum_{j=1}^m P(x_j) H(Y|X = x_j), \quad (2.5)$$

де

$P(x_j)$ – ймовірність того, що ознака X приймає значення x_j ;

$H(Y|X = x_j)$ – ентропія класу Y за умови $X = x_j$, що обчислюється за формулою

$$H(Y|X = x_j) = -\sum_{i=1}^k P(C_i|X = x_j) \log_2 P(C_i|X = x_j). \quad (2.6)$$

Поєднуючи формули (2.5) та (2.6), отримуємо загальну формулу умовної ентропії:

$$H(Y|X) = -\sum_{j=1}^m P(x_j) \sum_{i=1}^k P(C_i|X = x_j) \log_2 P(C_i|X = x_j). \quad (2.7)$$

Для практичного розрахунку умовної ентропії слід виконати такі дії:

1. Обчислення умовних ймовірностей $P(C_i|X = x_j)$: для кожного значення класу C_i та кожного значення ознаки x_j визначаємо частоту, з якою клас C_i зустрічається при значенні $X = x_j$. Це означає підрахунок кількості рядків, де одночасно $Y = C_i$ та $X = x_j$.

2. Обчислення ентропії для кожного значення ознаки: застосовуємо формулу (2.6) для обчислення ентропії $H(Y|X = x_j)$ для кожного значення x_j .

3. Зважування та підсумовування: отримані значення ентропії зважуємо на ймовірності кожного значення ознаки $P(x_j)$ і підсумовуємо їх, щоб отримати загальну умовну ентропію $H(Y|X)$ за формулою (2.7).

Взаємна інформація, або інформаційний вигравш $G(Y, X)$ показує, наскільки знання про ознаку X зменшує невизначеність щодо класу Y . Вона обчислюється як різниця між ентропією $H(Y)$ та умовною ентропією $H(Y|X)$:

$$G(Y, X) = H(Y) - H(Y|X). \quad (2.8)$$

Наведемо приклад обчислення умовної ентропії та інформаційного виграшу. Припустимо, ми маємо набір даних з однією ознакою X , яка може набувати значень x_1 та x_2 , та бінарним класом Y , що має два значення C_1 та C_2 . Інформація про розподіл цих значень подана в таблиці 2.1.

Загальний розподіл класів:

– всього 100 рядків, з яких 60 належать до класу C_1 і 40 – до класу C_2 .

Розподіл значень ознаки X :

- значення x_1 зустрічається у 70 рядках, з них 50 належать до C_1 , а 20 – до C_2 ;
- значення x_2 зустрічається у 30 рядках, з них 10 належать до C_1 , а 20 – до C_2 .

Таблиця 2.1

Набір даних прикладу

	C_1	C_2	Разом
x_1	50	20	70
x_2	10	20	30
Разом	60	40	100

Обчислимо ентропію $H(Y)$ за формулою (2.4):

$$P(C_1) = \frac{60}{100} = 0.6, \quad P(C_2) = \frac{40}{100} = 0.4$$

$$H(Y) = -(0.6 \log_2 0.6 + 0.4 \log_2 0.4) \approx 0.97$$

Обчислимо ентропію для кожного значення ознаки за формулою (2.6):

$$P(C_1|X = x_1) = \frac{50}{70} = \frac{5}{7}, \quad P(C_2|X = x_1) = \frac{20}{70} = \frac{2}{7},$$

$$H(Y|X = x_1) = -\left(\frac{5}{7} \log_2 \frac{5}{7} + \frac{2}{7} \log_2 \frac{2}{7}\right) \approx 0.86,$$

$$P(C_1|X = x_2) = \frac{10}{30} = \frac{1}{3}, \quad P(C_2|X = x_2) = \frac{20}{30} = \frac{2}{3},$$

$$H(Y|X = x_2) = -\left(\frac{1}{3} \log_2 \frac{1}{3} + \frac{2}{3} \log_2 \frac{2}{3}\right) \approx 0.92,$$

Загальна умовна ентропія $H(Y|X)$ обчислюється за формулою (2.7):

$$H(Y|X) = \frac{70}{100} \times 0.86 + \frac{30}{100} \times 0.92 \approx 0.88.$$

Розрахуємо взаємну інформацію за формулою (2.8):

$$G(Y, X) = H(Y) - H(Y|X) = 0.97 - 0.88 = 0.09.$$

Основна ідея методу [108] полягає тому, щоб для імпутовання пропущеного значення ознаки X вибрати таке x_j з можливих варіантів x_1, x_2, \dots, x_m , яке максимізує взаємну інформацію $G(Y, X) \rightarrow \max$. Оскільки

$H(Y)$ не залежить від X , то це еквівалентно мінімізації умовної ентропії. Отже, необхідно знайти x_j , при якому $H(Y|X) \rightarrow \min$.

Пропонується виконувати імпутування кожної ознаки окремо за наступним алгоритмом, який є реалізацією описаного методу. Розглянемо спочатку випадок категоріальної ознаки.

Алгоритм 2.5 – Імпутування категоріальної ознаки

1. Розглянемо піднабір даних D , що складається з ознаки X та цільового стовпця Y . Поділяємо дані на дві множини: повну D^f (містить дані у всіх ознаках) та неповну D^{nf} (рядки, що містять пропуски).

2. Беремо перший рядок з D^{nf} і виконуємо крок підбору значення для імпутації.

3. Крок підбору значення: для кожного можливого значення x_j , де $j \in [1, \dots, m]$ розраховуємо умовну ентропію на множині D^f , доповненій значенням x_j .

4. Вибираємо в якості значення, що імпутується, таке значення x_j , яке мінімізує умовну ентропію.

5. Оновлюємо набір даних, заповнивши пропущене значення у вибраному рядку.

6. Повторюємо кроки 2 – 5, поки в множині D^{nf} залишаються пропуски.

Цей алгоритм можна модифікувати шляхом проведення кількох ітерацій підбору значень для імпутації, поки загальна умовна ентропія продовжує зменшуватись.

Фактично виявляється, що для пропущеного значення класу C_i оптимальне значення x_j є тим, яке максимізує умовну ймовірність $P(C_i|X = x_j)$ у заповненому наборі даних.

Алгоритм 2.6 – Імпутування категоріальної ознаки ітераційно

1. Встановлюємо максимально допустиму кількість ітерацій.

2. Встановлюємо номер ітерації в 1.
3. Встановлюємо поточне значення умовної ентропії у заздалегідь задане велике число, наприклад, 10000.
4. Якщо номер ітерації не перевершує максимально допустиму кількість, продовжуємо алгоритм, інакше завершуємо його.
5. Зберігаємо поточний вміст ознаки X .
6. Виконуємо алгоритм 2.5 для підбору значень.
7. Обчислюємо умовну ентропію після імпутації.
8. Якщо значення умовної ентропії зменшилось порівняно з попереднім значенням, переходимо до кроку 9, інакше – до кроку 11.
9. Оновлюємо набір даних, приймаючи всі виконані імпутації.
10. Збільшуємо номер ітерації та переходимо до кроку 4.
11. Якщо умовна ентропія не зменшилась порівняно з попереднім значенням, повертаємося до попередньої версії імпутації і завершуємо алгоритм.

Для обох алгоритмів наведено UML-діаграми діяльності, представлені на рисунку 2.6, що візуально ілюструють їхню роботу.

З'ясувалося, що для збіжності умовної ентропії у проведених в роботі експериментах зазвичай достатньо 3–4 ітерацій, після чого її значення більше не зменшується.

Якщо ознака є якісною порядковою, пропонується виконати крок її перетворення у кількісну, застосовуючи методи з підрозділу 2.1. Цей підхід дозволяє зберегти інформацію про пропуски під час перетворення всіх якісних ознак на кількісні, після чого ознака буде розглядатись як дискретна.

У разі неперервної ознаки, задачу можна спростити до дискретної шляхом її дискретизації.

Пропонується визначати кількість проміжків як кількість класів варіаційного ряду за формулою:

$$M = 1 + 3.32 \cdot \lg(n), \quad (2.9)$$

де n – кількість рядків у множині повних даних D^f .

Це перетворення дозволяє працювати з ознакою як із дискретною величиною. Для зворотного перетворення, коли необхідно відновити значення з імпутованого інтервалу, пропонується генерувати випадкову величину з нормального розподілу $N(m; \sigma)$, де m – середнє значення ознаки X з D^f , σ – середньо-квадратичне відхилення цієї ознаки.

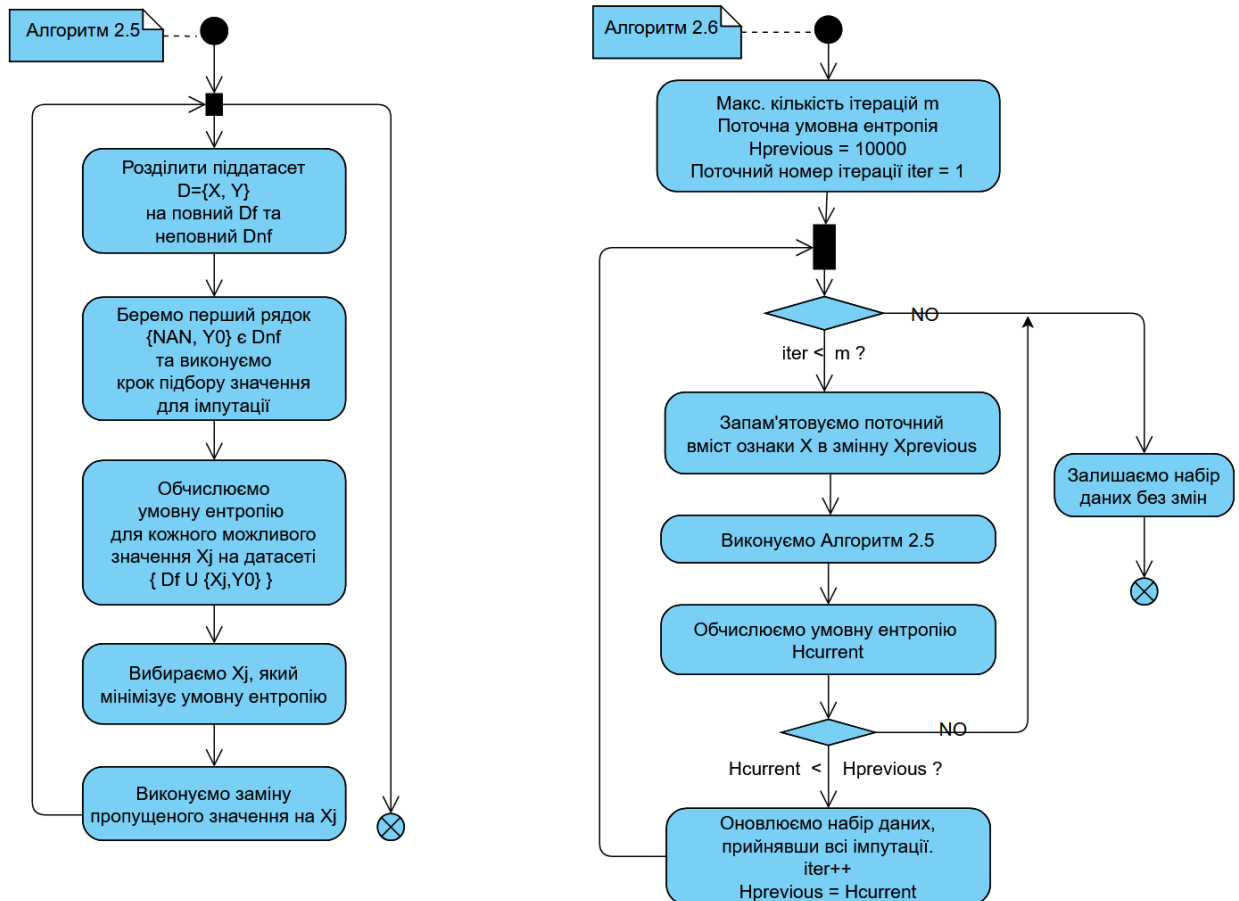


Рисунок 2.6 – Діаграми діяльності UML алгоритмів 2.5 та 2.6

Цей алгоритм вимагає значних обчислювальних ресурсів, але його продуктивність можна підвищити за допомогою паралельних обчислень [116]. Оскільки імпутація кожної ознаки може виконуватися незалежно від інших, обробку можна розподілити між окремими потоками. Крім того, ефективність можна покращити завдяки оптимізаціям, характерним для конкретних мов програмування, наприклад, векторизації обчислень [117] у Python. Проведено рефакторинг алгоритмів та розроблено клас `EntropyImputer` – імп'ютер, інтегрований в екосистему `scikit-learn` на мові Python.

2.5 Гібридний метод імпутування HybridRegrEntropyImputer

Аналіз недоліків попереднього методу показав, що втрата якості та підвищене використання обчислювальних ресурсів виникають переважно під час імпутації кількісних ознак. Зважаючи на це, було запропоновано створити новий гібридний метод HybridRegrEntropyImputer [119], який поєднує регресійний і ентропійний підходи.

Згідно з концепцією цього імп'ютера, кількісні ознаки заповнюються за допомогою RegrImputer, тоді як якісні – за допомогою EntropyImputer. Такий підхід дозволив покращити якість імпутації та значно скоротити час обчислень, що робить цей імп'ютер конкурентним серед інших запропонованих методів за всіма розглянутими показниками.

Далі проведено рефакторинг алгоритму та створено клас HybridRegrEntropyImputer у форматі, сумісному з архітектурою scikit-learn для мови Python.

2.6 Метод імпутування CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками

Якщо під час початкового статистичного аналізу даних виявляється стохастичний зв'язок між ознаками, це можна використати для ефективнішого заповнення пропущених значень. Зокрема, пропуски у даних можуть бути імпутовані на основі підібраної регресійної залежності між показниками [109].

У цьому випадку будемо застосовувати регресійний аналіз, використовуючи як лінійну регресію, так і 7 моделей квазілінійних залежностей, які можна привести до лінійної моделі шляхом відповідного перетворення.

Розглядаються такі 8 моделей регресії:

1. Лінійна модель регресії $y = a + b * x$
2. Модель $y = a + b * \ln(x)$
3. Модель $y = \exp(a + b * x)$

4. Модель $y = \exp(a + b * \ln(x))$
5. Модель $y = \ln(a + b * x)$
6. Модель $y = a + b/x$
7. Модель $y = a + b * \exp(x)$
8. Модель $y = \sqrt{a + b * x}$

Далі наведемо алгоритм, що описує процес імпутування на основі регресійного аналізу.

Алгоритм 2.7 – Алгоритм імпутування на основі регресійного аналізу

1. Вибір підмножини даних: Розглядається підмножина датасету $D\{X, Y\}$, де X, Y – дві ознаки з потенційним зв'язком.
2. Фільтрація даних: Видаляються рядки, де значення X та Y одночасно відсутні.
3. Перевірка пропущених значень: Якщо пропущених значень не виявлено, алгоритм завершується.
4. Аналіз відсутніх даних: Визначається кількість рядків, де значення Y відсутнє, але X наявне. Якщо таких рядків немає, переходять до кроку 11.
5. Пошук залежності: Визначається стохастична залежність $Y = f(X) + \varepsilon$.
6. Обчислення кореляції: Розраховується коефіцієнт кореляції Пірсона та перевіряється його значущість.
7. Альтернативне імпутування: Якщо кореляція не є значущою, можна використати простий метод імпутації (SimpleImputer) і завершити алгоритм.
8. Моделювання залежності: Будуються 8 моделей регресії (лінійна та квазілінійні).
9. Оцінка моделей: Обирається модель f з найменшим середнім квадратичним відхиленням (MSE).
10. Імпутація значень: Заповнюються пропуски у Y за формулою $Y = f(X)$.

11. Зворотний аналіз: Змінюються місцями ознаки X та Y , тобто аналізується залежність $X = f(Y) + \varepsilon$.

12. Повторення циклу: Повертаємося до кроку 3.

UML-діаграму діяльності алгоритму 2.7 представлено на рисунку 2.7 (а).

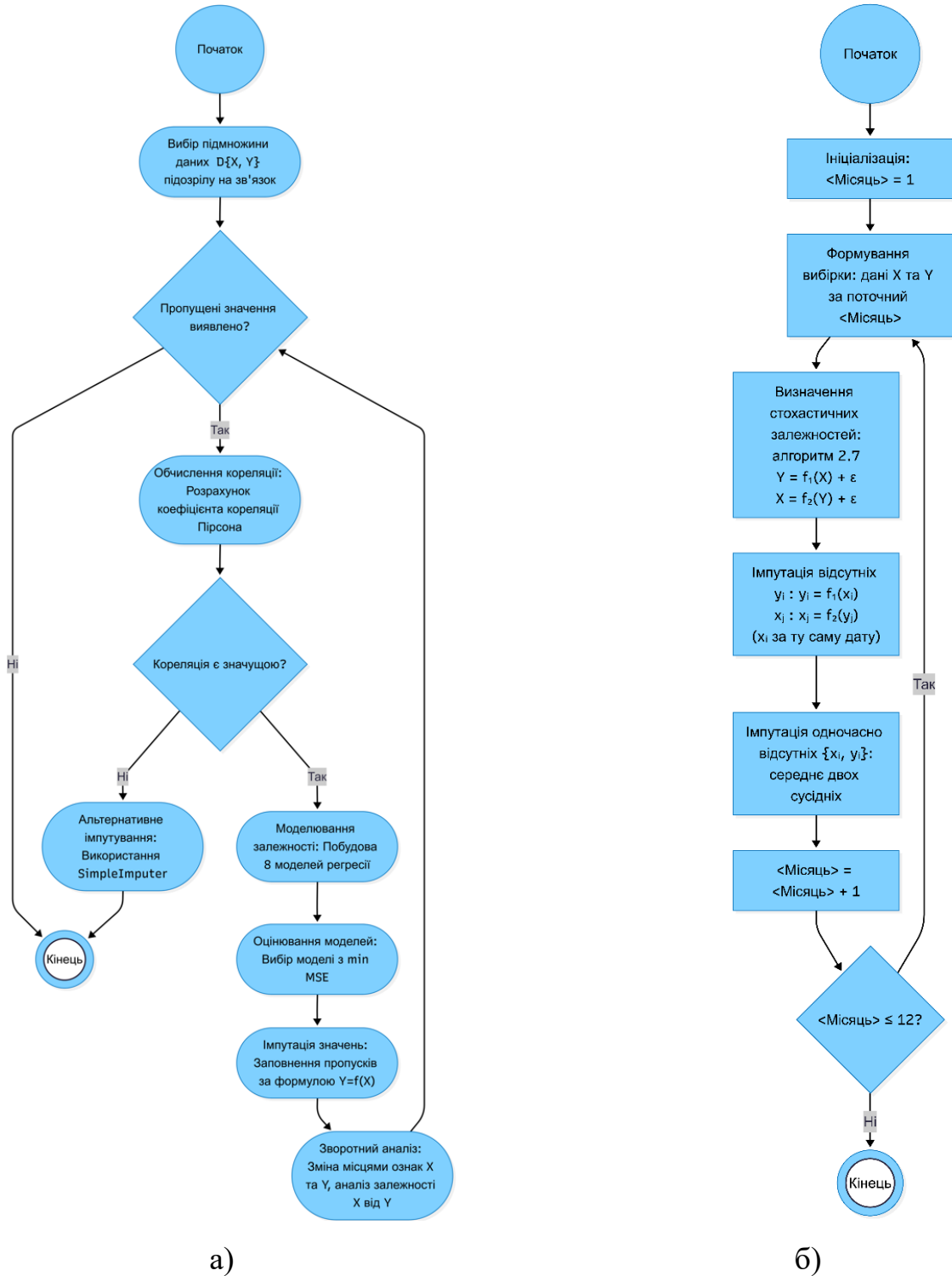


Рисунок 2.7 – UML-діаграма діяльності алгоритму 2.7 (а) та 2.8 (б)

Далі проведено рефакторінг алгоритму та розроблено клас `CorrelationImputer` – ім'ютер на мові програмування Python згідно з архітектурними патернами бібліотеки `scikit-learn`.

Імпутування пропусків у даних часових рядів з сезонною складовою.

Якщо мова йде про часові ряди з наявною сезонною складовою, то метод `CorrelationImputer` може бути використаний наступним чином.

Далі представлено нову методику заповнення пропусків у даних для випадку двох часових рядів з сезонною складовою. Перетворимо часові ряди датасету на випадкові величини за ознакою X та ознакою Y (наприклад, рівень води та витрати води), взявши всі дані цих ознак за конкретний місяць року. Тобто з набору даних відбираємо інформацію про ознаки X та Y за обраний місяць усіх років, що формує сукупні дані для одного місяця без прив'язки до дат.

Алгоритм 2.8 – Помісячне імпутування пропусків у часових рядах датасету `hydro_monitoring` методом `CorrelationImputer`

1. $\langle \text{Місяць} \rangle = 1$ (січень)
2. Формуємо вибірки – дані про ознаки X та Y за $\langle \text{Місяць} \rangle$
3. Визначаємо найкращі стохастичні залежності за алгоритмом 2.7:
$$Y = f_1(X) + \varepsilon$$
 та
$$X = f_2(Y) + \varepsilon.$$
4. Відсутні y_i в межах місяця $\langle \text{Місяць} \rangle$ будь-кого року знаходимо за формулою $y_i = f_1(x_i)$, де x_i береться за ту саму дату.
5. Відсутні x_j знаходимо за формулою $x_j = f_2(y_j)$ аналогічно кроку 5.
6. Для відсутніх $\{x_i, y_i\}$ одночасно, замінюємо їх на відповідне середнє значення двох сусідніх наявних.
7. $\langle \text{Місяць} \rangle = \langle \text{Місяць} \rangle + 1$
8. Якщо $\langle \text{Місяць} \rangle \leq 12$, переходимо на крок 3, інакше – завершуємо алгоритм.

На рисунку 2.7 (б) зображена UML-діаграма діяльності алгоритму.

2.7 Висновок до розділу 2

В розділі 2 наведено розроблені методи перетворення порядкових категоріальних ознак на кількісні зі збереженням інформації про пропуски даних з деталізацією у вигляді лістингів алгоритмів: `IgnoreNaNLabelEncoder` на основі `LabelEncoder` з бібліотеки `scikit-learn` на мові програмування Python (лістинг 2.1) та `IgnoreNaNFrequentEncoder` (лістинг 2.2), який додатково враховує частоту конкретних значень. Ці алгоритми надають можливість зворотного перетворення до початкових даних також зі збереженням інформації про пропущені дані.

Представлено розроблені методи імпутування пропусків у даних:

1. Уніфікований `UnifiedClassRegrImputer`, який поєднує алгоритми 2.1–2.3, і оснований на застосуванні класифікатора для якісних ознак та регресору – для кількісних, враховує патерни у даних та надає можливість змінного порядку обходу ознак залежно від кількості пропусків.
2. Імп'ютер `RegrImputer` на основі застосування регресору з корегуванням значення для якісних ознак (алгоритм 2.4).
3. Метод імпутування `EntropyImputer` на основі ентропійного підходу (алгоритми 2.5, 2.6).
4. Гібридний метод імпутування `HybridRegrEntropyImputer`, що поєднує ентропійний та регресійний підходи.
5. Метод імпутування `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками (алгоритм 2.7).
6. Метод помісячного імпутування пропусків у часових рядах датасету з використанням методу `CorrelationImputer` (алгоритм 2.8).

РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МЕТОДІВ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ

В цьому розділі представлено опис програмного забезпечення методів імпутування пропусків у даних. Підрозділ 3.1 присвячений опису розроблених класів імпультерів відповідно до архітектурних принципів бібліотеки `scikit-learn` мови програмування Python. Підрозділ 3.2 присвячений інженерним технікам щодо оптимізації коду на мові програмування Python. В підрозділі 3.3 порівняно багатопроцесну та багатопоточну реалізацій ентропійного підходу для імпутування пропусків у даних. Висновки до розділу наведено в підрозділі 3.4.

3.1 Розроблення методів для імпутування пропусків у даних відповідно до архітектурних принципів бібліотеки `scikit-learn` Python

Ця частина дослідження спрямована на розробку нових класів для імпутування пропущених даних, які можна легко інтегрувати в стандартні процеси обробки даних, зокрема в аналітичні конвеєри (pipeline) бібліотеки `scikit-learn`. У таких конвеєрах поєднуються різні трансформери, наприклад, масштабування, нормалізація, імпутація пропусків, а також моделі машинного навчання, такі як класифікація чи прогнозування.

Головна мета – забезпечити гнучкість, масштабованість та сумісність з існуючими компонентами `scikit-learn`, зокрема зі `scaler`-ами (наприклад, `MinMaxScaler`) для масштабування даних та класифікаторами (наприклад, `RandomForestClassifier`) для класифікації.

У другому розділі описано розроблені методи та алгоритми імпутування, а наступні етапи включають рефакторинг коду та адаптацію попередніх розробок до архітектурних вимог `scikit-learn` [89], щоб гарантувати їхню уніфіковану інтеграцію в стандартні процеси аналізу даних.

Згідно з архітектурою `scikit-learn`, всі трансформери, зокрема імпультери, повинні відповідати певним вимогам для забезпечення сумісності з

аналітичними конвеєрами (pipeline). Основні вимоги до нових класів імпультерів включають [118]:

1. Імпультери повинні наслідуватись від класів `BaseEstimator` і `TransformerMixin` з пакету `sklearn.base`, що дозволяє використовувати методи `fit`, `transform`, та `fit_transform` відповідно до архітектури `scikit-learn`.

2. Реалізація обов'язкових методів:

– `fit(X, y=None)`: обчислює параметри, необхідні для імпутування, на основі наданого датасету;

– `transform(X)`: виконує заповнення пропусків у даних X ;

– `fit_transform(X, y=None)`: поєднує виклики методів `fit` і `transform`.

3. Підтримка параметрів:

– `missing_values`: вказує значення, які слід замінити (наприклад, `np.nan`, `None` або спеціальне значення);

– `add_indicator` (опційно): додає бінарний індикатор для позначення місць пропусків, що іноді корисно для моделей машинного навчання;

– свої власні параметри, характерні для методу.

4. Сумісність з pipeline: нові імпультери повинні інтегруватися через pipeline із іншими трансформерами (наприклад, `MinMaxScaler` для масштабування ознак) і моделями (наприклад, `RandomForestClassifier` для класифікації), забезпечуючи комплексну обробку даних та моделювання.

Розроблено класи `UnifiedClassRegrImputer` (відповідно до методу з підрозділу 2.2), `RegrImputer` (відповідно до методу з підрозділу 2.3), `EntropyImputer` (згідно з методом з підрозділу 2.4), `HybridRegrEntropyImputer` (з підрозділу 2.5) та `CorrelationImputer` (відповідно до методу з підрозділу 2.6). Реалізація цих класів наведена у додатку Б. Оскільки розроблені класи повністю відповідають архітектурним вимогам `scikit-learn`, вони сумісні зі стандартними інструментами бібліотеки та можуть бути легко інтегровані в аналітичні конвеєри (pipeline) разом з іншими трансформерами та моделями. Використання pipeline дозволяє автоматизувати послідовність етапів обробки

даних – від попередньої трансформації до моделювання, що особливо зручно при роботі з великими наборами даних.

Наприклад, для задачі класифікації, якщо потрібно виконати імпутацію пропусків (за допомогою `RegrImputer`), масштабування ознак та класифікацію (наприклад, за допомогою `RandomForestClassifier`), код може виглядати так (лістинг 3.1):

Лістинг 3.1 – Код конвеєру

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from zo_imputers import RegrImputer # Новий імп'ютер
from sklearn.ensemble import RandomForestClassifier

# Розділення даних на X та y
X = data.drop(target, axis=1)
y = data[target]
# Розділення даних на тренувальний та тестовий набір,
# train та test
X_train, X_test, y_train, y_test =
    train_test_split(X, y,
                    test_size=0.3,
                    random_state=42)
# Створення pipeline з імпутацією, масштабуванням
# та класифікацією
pipeline = Pipeline([
    ('imputer', RegrImputer(measure='value')),
    ('scaler', MinMaxScaler()),
    ('classifier', RandomForestClassifier(n_estimators=100))
])
# Класифікація: навчання, прогнозування та оцінка моделі
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Для порівняння наведемо також код зі звичайним запуском тих самих дій без застосування `pipeline` (лістинг 3.2).

Лістинг 3.2 – Код без використання конвеєру

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from zo_imputers import RegrImputer # Новий імп'ютер
```

```

# Імпутація
imputer = ReprImputer(measure='value')
data = imputer.fit_transform(data)
# Розділення даних на X та y
X = data.drop(target, axis=1)
y = data[target]
# Розділення даних на тренувальний та тестовий набір,
# train та test
X_train, X_test, y_train, y_test =
    train_test_split(X, y,
                    test_size=0.3,
                    random_state=42)

# Масштабування
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = RandomForestClassifier(n_estimators=100)
# Класифікація: навчання, прогнозування та оцінка моделі
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))

```

Можна помітити, наприклад, що імпутування потрібно виконувати перед розділенням даних. Масштабування виконано окремо для тестового та навчального наборів. Для автоматизації процесу, який включає різні методи імпутації, масштабування та класифікації, знадобилося б створити функцію.

Переваги підходу:

1. Модульність: дозволяє легко змінювати чи додавати нові компоненти у pipeline.
2. Масштабованість: підходить для роботи з великими наборами даних.
3. Гнучкість: забезпечує простоту у виборі методів імпутації та моделей класифікації.

Таким чином, запропоновані методи імпутації пропусків інтегруються у архітектуру scikit-learn, відповідають усім вимогам сумісності з pipeline, підтримують масштабування та демонструють високу гнучкість при вирішенні задач класифікації.

3.2 Векторизація обчислень для оптимізації коду на мові програмування Python

Python – одна з найпоширеніших мов програмування, особливо в аналізі даних, машинному навчанні та наукових обчисленнях. Проте, незважаючи на її гнучкість та універсальність, продуктивність стандартного Python-коду може бути недостатньою для обробки великих обсягів даних. Одним з найефективніших способів прискорення обчислень є векторизація [117] – техніка, яка дозволяє обробляти цілі масиви даних одночасно, уникаючи повільних циклів.

Мета цього дослідження – оцінити, як векторизація може покращити продуктивність та читабельність коду, а також продемонструвати її переваги на практичних прикладах з власного коду.

Векторизація – це інженерний підхід, який передбачає виконання операцій над векторами або масивами даних за допомогою оптимізованих функцій, написаних на мовах низького рівня, наприклад, C або Rust. Такі функції можуть обробляти цілі структури даних, уникаючи явних циклів *for* [91]. Прикладами бібліотек, написаних на C або Rust, та які надають ефективні методи для роботи з масивами, є NumPy, Pandas чи Numba.

Переваги векторизації включають:

1. Збільшення швидкодії: операції над масивами виконуються на низькому рівні (зазвичай у машинному коді), що значно прискорює обчислення порівняно з традиційними циклами.
2. Лаконічність та читабельність: відмова від вкладених циклів робить код коротшим, чистішим і зрозумілішим.
3. Ефективне використання пам'яті: векторизовані операції зменшують кількість звернень до пам'яті, покращуючи загальну продуктивність програми.

Застосування готових векторизованих функцій з бібліотек. Спочатку проаналізуємо вбудовані векторизовані рішення, доступні в бібліотеках NumPy

та Pandas. Ці інструменти дозволяють обробляти цілі масиви даних одночасно, без необхідності використання циклів.

Порівняємо традиційний підхід (з використанням циклів) та векторизацію, оцінюючи їх ефективність за трьома критеріями: 1) швидкість обчислень; 2) компактність коду; 3) його зрозумілість.

Як практичний приклад розглянемо задачу, що виникла під час розробки алгоритмів імпутування – це допоможе продемонструвати реальні переваги векторизації [107].

Для опрацювання патерну d) з алгоритму 2.2 (наявні m рядків, де є пропуск тільки в одній ознаці) потрібно було знайти такі ознаки, тобто потрібно було написати функцію, що повертає імена стовпців датасету, для яких є рядки з пропуском лише в цьому стовпці. Розглянемо дві реалізації цієї функції. В лістингу 3.3а наведено реалізацію із застосуванням циклів. В лістингу 3.3б ця функція заснована на векторизації обчислень.

Лістинг 3.3а – Реалізація з використанням циклів

```
def get_col_names_with_one_nan_in_row_loop(data):
    df = data.copy()
    # Додаємо стовпець 'count', де кожен елемент - це
    # кількість пропусків (NaN) у рядку
    counts = []
    for i in range(len(df)):
        count_nan = 0
        for col in df.columns:
            if pd.isnull(df.iloc[i][col]):
                count_nan += 1
        counts.append(count_nan)
    df['count'] = counts
    # Додаємо стовпець 'nans', де кожен елемент - це
    # назви стовпців через кому, що містять NaN у рядку
    nans = []
    for i in range(len(df)):
        nan_columns = []
        for col in df.columns:
            if pd.isnull(df.iloc[i][col]):
                nan_columns.append(col)
        nans.append(",".join(nan_columns))
    df['nans'] = nans
    # Створюємо список унікальних назв стовпців,
```

```

# які мають лише один пропуск у рядку саме в цьому стовпці
d = []
for i in range(len(df)):
    if df.iloc[i]['count'] == 1:
        nan_columns = df.iloc[i]['nans']
        if nan_columns:
            d.extend(nan_columns.split(", "))
d = np.unique(d)
return d

```

Лістинг 3.36 – Реалізація з використанням векторизації

```

def get_col_names_with_one_nan_in_row_vect(data):
    df = data.copy()
    # Додаємо стовпець 'count', де кожен елемент - це
    # кількість пропусків (NaN) у рядку
    df['count'] = df.isnull().sum(axis=1)
    # Додаємо стовпець 'nans', де кожен елемент - це
    # назви стовпців через кому, що містять NaN у рядку
    df['nans'] = (df.isnull() @ (df.columns + ",")).str[:-1]
    # Створюємо список унікальних назв стовпців,
    # які мають лише один пропуск у рядку саме в цьому стовпці
    d = df[df['count'] == 1]['nans'].tolist()
    d = np.unique(d)
    return d

```

Результати використання векторизації. Порівняємо лістинги 3.3а та 3.3б. Лістинг 3.3б є значно компактнішим порівняно з лістингом 3.3а, що позитивно впливає на читабельність і зрозумілість коду. Крім того, результати експериментів із реальними наборами даних демонструють значну перевагу в швидкодії векторизованого підходу.

Для аналізу використовували два набори даних: UCI Heart Disease Data (UCI HDD) [13, 14] та Framingham Heart Study (Framingham FHS) [15, 16]. В таблиці 3.1 наведено розміри наборів даних та середній час виконання методів.

Таблиця 3.1

Порівняння швидкодії двох реалізацій методу

Набір даних	Кількість рядків	Кількість стовпців	Час виконання, с		Швидкість ↑
			Метод з використанням циклів	Метод з векторизацією обчислень	
UCI HDD	920	16	2.040	0.0038	540
Framingham FHS	4240	18	10.328	0.0060	1730

На наборі UCI HDD швидкість виконання зростає у 540 разів при використанні векторизації, а на наборі Framingham FHS продуктивність покращилась у 1730 разів.

Окремо розглянемо наступний рядок з лістингу 3.3б:

```
df['nans'] = (df.isnull() @ (df.columns + ",")).str[:-1]
```

Цей код використовує матричну операцію `@` з бібліотеки Pandas для створення рядків, які містять назви стовпців з пропущеними значеннями (NaN) у кожному рядку DataFrame. Розберемо роботу коду крок за кроком.

1. `df.isnull()`:

Цей метод створює булевий DataFrame, що відповідає розмірам вихідного df, де значення *True* відповідають пропущеним значенням у df (NaN), а *False* – заповненим. Таким чином, отримуємо маску пропусків, яка допомагає ідентифікувати місця з відсутніми даними в оригінальному DataFrame.

df	A	B	C
0	1	3	NaN
1	NaN	4	NaN
2	2	NaN	5

df.isnull()	A	B	C
0	False	False	True
1	True	False	True
2	False	True	False

Рисунок 3.1 – Вихідний DataFrame df (ліворуч) та df.isnull() (праворуч)

2. `df.columns + ", "`:

Створюється масив з назвами всіх стовпців, до кожного елемента якого додається кома. В нашому прикладі буде такий результат

```
Index(['A,', 'B,', 'C,'], dtype='object')
```

3. `df.isnull() @ (df.columns + ", ")`:

У цьому виразі оператор `@` реалізує матричне множення між булевою матрицею пропусків (`df.isnull()`) та масивом назв стовпців з доданими комами (`df.columns + ", "`).

Як це працює:

- кожен рядок булевої матриці (True – пропуск, False – заповнене значення) поелементно множиться на відповідні назви стовпців з комами;
- в результаті для кожного рядка об'єднуються назви стовпців, де є NaN, утворюючи рядок з переліком стовпців з пропусками, розділених комами.

Результат: для кожного рядка вихідного DataFrame формується рядок з назвами стовпців, у яких відсутні дані. Наприклад, якщо в рядку пропуски є в стовпцях A та C, результат буде: «A,C,». В нашому прикладі отримаємо

```
0    C,  
1    A, C,  
2    B,
```

Рисунок 3.2 – Результат операції `df.isnull() @ (df.columns + ",")`

Це означає, що в першому рядку тільки стовець C містить значення NaN, у другому – стовпці A і C, а в третьому – стовець B.

4. `.str[:-1]`

Ця операція видаляє останню кому в кожному рядку.

Таким чином, за допомогою швидких векторизованих операцій з Pandas, наведений код формує новий текстовий стовець, який для кожного рядка містить перелік назв стовпців з пропущеними даними в цьому рядку, розділених комами.

Векторизація власних функцій. Існує техніка перетворення власних функцій на векторизовані за допомогою `np.vectorize` з бібліотеки NumPy. Цей інструмент дозволяє застосовувати функцію до кожного елемента масиву, імітуючи векторизовану поведінку.

Важливо зазначити, що `np.vectorize` не прискорює обчислення, а лише автоматизує виклик функції для кожного елемента, фактично приховуючи цикл *for*. Це не підвищує продуктивність, але покращує читабельність та компактність коду.

У лістингу 3.4 наведено приклад використання `np.vectorize` у власному коді.

Лістинг 3.4 – Фрагмент коду з `np.vectorize`

```
Y_test = self.regressor.predict(X_test)

if is_column_type_categorical(df[i].unique()):
    d = df[i].value_counts().to_dict()
    if self.measure == 'value':
        Y_test = np.vectorize(lambda x:
                               correct_value(d, x))(Y_test)
    else:
        Y_test = np.vectorize(lambda x:
                               correct_value_weight(d, x))(Y_test)

df.loc[index_nan, i] = Y_test
```

Ця техніка дозволяє уникнути явних циклів, забезпечуючи корекцію значень відразу для всіх елементів стовпця `DataFrame`.

Висновки щодо `np.vectorize`:

1. Зручність: код стає простішим і лаконічнішим, оскільки не потрібно писати цикли вручну.
2. Не прискорює обчислення: незважаючи на назву, `np.vectorize` не оптимізує швидкодію – це лише зручна обгортка над циклом *for*, яка автоматизує застосування функції до кожного елемента.
3. Призначення: використовується для зручного застосування довільних функцій до масивів, але не для оптимізації продуктивності.

Досліджено векторизацію з двох точок зору:

- підвищення швидкодії (завдяки оптимізованим бібліотекам);
- покращення компактності та читабельності коду.

На прикладі роботи власного коду для імпутування пропущених значень продемонстровано значні переваги векторизації як для прискорення обчислень, так і для покращення читабельності коду.

Отже, векторизація є потужним інструментом оптимізації Python-коду, особливо при роботі з великими обсягами даних та складними обчисленнями. Вона суттєво скорочує час виконання програм і покращує їхню читабельність. Використання бібліотек, таких як NumPy і Pandas, дозволяє ефективно

реалізувати векторизовані операції, що є критично важливим у сучасній науці про дані та інженерних застосунках.

3.3 Порівняння багатопроцесної та багатопоточної реалізацій ентропійного методу для імпутації пропусків у Python

Це дослідження зосереджується на аналізі ефективності багатопроцесного та багатопоточного підходів у реалізації ентропійного алгоритму імпутації пропусків у даних на мові Python [116]. Мета роботи – оцінити вплив різних методів паралелізації на швидкість виконання завдань імпутації. Використання таких підходів є особливо важливим при обробці великих обсягів даних, хоча Python має певні обмеження через специфіку свого інтерпретатора.

Оскільки Python є інтерпретованою мовою, код виконується через програму-інтерпретатор. Хоча багатопоточність зазвичай вважається ключовим методом оптимізації обчислень, у Python її ефективність обмежена через наявність глобального блокування інтерпретатора (GIL) [92]. *GIL (Global Interpreter Lock)* – це м'ютекс, який дозволяє одночасно працювати з об'єктами Python лише тому потоку, що отримав цей блокувальний механізм. Інтерпретатор Python періодично переключає потоки, створюючи ілюзію паралельного виконання. Цей механізм забезпечує цілісність і безпеку даних, але значно ускладнює використання кількох ядер процесора для паралельних обчислень. З виходом Python 3.2 була введена часткова оптимізація GIL, яка покращила продуктивність у деяких випадках, але не усунула головного обмеження. Через це потоки у Python краще підходять для задач, що пов'язані з очікуванням введення-виведення, ніж для інтенсивних обчислень.

Для подолання обмежень GIL рекомендується використовувати багатопроцесний підхід. На відміну від потоків, кожен процес має власний інтерпретатор Python і власний GIL, що дає змогу повноцінно використовувати всі ядра процесора. Такі бібліотеки, як *concurrent.futures*, надають зручний інтерфейс для роботи з пулами потоків (*ThreadPoolExecutor*) і процесів

(*ProcessPoolExecutor*), дозволяючи адаптувати методи паралелізації до конкретних потреб.

Дослідження проводилось на Python 3.12 із використанням датасету UCI Heart Disease Data [14], який містить дані з кардіологічних центрів США, Угорщини та Швейцарії. Для оцінки ефективності методів штучно створювались пропуски різного рівня: 10%, 20%, 30% і 40%, як описано в нашій роботі [107]. Імпутація виконувалась за ентропійним підходом трьома реалізаціями: послідовною, багатопотоковою і багатопроесною. Виконано 10 ітерацій генерування пропусків в даних та імпутацій. Обчислено середню помилку імпутації (метрика RMSE) для кожного обсягу пропусків та середній загальний час імпутування.

Імпутація кожної ознаки проводилась незалежно, що дозволяло розподілити завдання між потоками або процесами без ризику перегонів даних (race conditions). У якості інструментів використовувались *ThreadPoolExecutor* та *ProcessPoolExecutor* [93].

Тести проводились на процесорі Intel i7-3770 із підтримкою Hyper-Threading, який забезпечує 4 фізичних і 8 логічних ядер. Для багатопроесного підходу було протестовано конфігурації з різною кількістю процесів: 4, 8 та варіант за замовчуванням (кількість процесів визначається системою). Результати експериментів, наведені у таблиці 3.2, підтверджують переваги багатопроесної реалізації з точки зору продуктивності.

Таблиця 3.2

Порівняння ефективності різних реалізацій ентропійного методу імпутування пропусків у даних

Реалізація	10%	20%	30%	40%	Час, с
Послідовний підхід	6.623	7.734	9.791	9.920	20.876
Багатопоточний підхід	6.596	7.466	9.802	9.730	22.014
Багатопроесний підхід, за замовчуванням	6.552	7.384	9.704	10.236	17.613
Багатопроесний підхід, 4 робочі процеси	6.644	7.502	9.894	9.904	15.487
Багатопроесний підхід, 8 робочих процесів	6.570	7.564	9.602	9.850	17.542

В стовпцях 10%, 20%, 30% та 40% наведено значення метрики RMSE для відповідного обсягу пропусків.

Результати демонструють, що різниця в середніх значеннях похибки імпутації між обраними підходами практично відсутня. Це пояснюється тим, що оптимізація була спрямована на підвищення швидкодії обчислень, а не на поліпшення точності. Порівняння послідовного та багатопоточного підходів показало, що багатопоточність у Python не забезпечує приросту швидкодії через обмеження, пов'язані з GIL, та може навіть погіршити продуктивність. Проте багатопроцесний підхід демонструє скорочення часу виконання обчислень. Дослідження підтвердило, що максимального покращення швидкодії можна досягти, якщо кількість робочих процесів дорівнює кількості фізичних ядер процесора. Отже при оптимізації обчислень важливо враховувати апаратні характеристики системи.

Таким чином, для оптимізації обчислень у Python можна рекомендувати застосовувати векторизовані обчислення за допомогою бібліотек, таких як NumPy та Pandas. Слід уникати надлишкових операцій і зменшувати кількість багаторазових звернень до введення/виведення, оскільки це значно впливає на продуктивність. Використання глобальних змінних також варто мінімізувати, оскільки вони можуть уповільнювати виконання програми. Для ідентифікації найресурсомісткіших частин коду доцільно застосовувати профілювання, наприклад, за допомогою модуля cProfile.

3.4 Висновок до розділу 3

У цьому розділі наведено опис програмного забезпечення для методів імпутування пропусків у даних, реалізованих згідно до архітектурних принципів бібліотеки scikit-learn мовою програмування Python.

Оскільки розроблені класи відповідають архітектурним вимогам scikit-learn, їх використання аналогічне стандартним трансформерам, а також вони можуть інтегруватися в pipeline разом з іншими трансформерами та моделями.

На прикладах власного коду розглянуто інженерні техніки оптимізації коду на мові Python. Встановлено, що векторизація обчислень значно підвищує швидкодію методів.

Для оптимізації коду ім'ютера на основі ентропійного підходу застосовано багато процесну та багатопоточну обробку даних. Під час програмних експериментів з'ясовано, що тільки багато процесна реалізація забезпечує перевагу у швидкодії, причому необхідно правильно визначати кількість робочих процесів.

РОЗДІЛ 4. РЕЗУЛЬТАТИ РОБОТИ МЕТОДІВ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ

Цей розділ присвячений аналізу роботи запропонованих методів імпутування пропусків у даних. Розглядаються два популярних датасети з даними кардіологічних досліджень – для задачі класифікації (підрозділ 4.1) та датасет з даними гідрологічного моніторингу басейну ріки Дніпро – для задачі прогнозування часових рядів (підрозділ 4.2). Аналіз точності імпутування та швидкодії представлений у підрозділі 4.3 (тест типу 1). Підрозділ 4.4 присвячений аналізу впливу імпутування на точність моделі класифікації (тест типу 2). В підрозділі 4.5 розглядається вплив імпутування на точність моделі прогнозування (тест типу 3). В підрозділі 4.6 аналізується умовна ентропія до та після імпутування (тест типу 4). В підрозділі 4.7 продемонстровано запуск методів у конвейері (pipeline). Висновок до розділу 4 наведений у підрозділі 4.8.

4.1 Опис датасетів для задач класифікації

Розглядаються два популярні відкриті набори даних, доступні на платформі Kaggle.com. Перший – це *UCI Heart Disease Data* [13, 14], сформований на основі даних із чотирьох кардіологічних центрів Угорщини, Швейцарії та США. Другий – *Framingham Heart Study* [15, 16], отриманий із активного кардіологічного дослідження мешканців Массачусетсу та Фремінгема.

Обидва набори даних містять пропуски й включають як кількісні, так і порядкові якісні ознаки, причому якісних ознак більше. У датасеті *UCI Heart Disease Data* частина пропусків пов'язана з об'єднанням кількох джерел, у яких не всі показники були представлені. Цей датасет має цільовий стовпець, що позначає стадії ішемічної хвороби серця від 0 (хвороба відсутня) до 4 (критична стадія). У цьому дослідженні він розглядається як задача бінарної класифікації (наявність або відсутність хвороби), оскільки в такій формі набір даних стає збалансованим.

Датасет *UCI Heart Disease Data* містить 920 рядків, 13 ознак та один цільовий стовпець. З 13 ознак 5 є кількісними, а 8 – порядковими якісними, з яких 2 мають бінарний характер. Первинний аналіз виявив пропуски: лише 33% записів є повними (299 екземплярів).

Датасет *Framingham Heart Study* складається з 4240 рядків, 14 ознак і цільового стовпця. Залишається 14 ознак: 8 кількісних і 6 якісних, із яких 5 є бінарними. Аналіз показав, що 86% екземплярів є повними (3658 рядків). Цей набір даних незбалансований: кількість екземплярів одного класу в п'ять разів перевищує кількість іншого. Для вирівнювання дисбалансу застосовано метод SMOTE [94].

4.2 Опис датасету для задачі прогнозування часових рядів

Датасет *hydro_monitoring* зібрано гідрометеорологічною службою України. Він містить багаторічні спостереження за режимом і ресурсами поверхневих вод басейну ріки Дніпро [23]. Дані були надані Науково-дослідним інститутом геології Дніпровського національного університету імені Олеся Гончара та науковою школою професора Шерстюк Н.П. [24, 25]. Дані представлено у вигляді щорічних звітів у форматах Word та Excel, що включають гідрометричні спостереження за рівнем і витратами води для рік, озер і водосховищ басейну Дніпра. Мережа охоплює 143 пункти спостереження. Діапазон даних охоплює 1991 – 2014 роки.

Неструктуровані дані були трансформовані Батурінець А.Г. у реляційну базу даних *hydro_monitoring* [26]. Спостереження виконувались щодня, формуючи часові ряди. Аналіз даних виявив пропуски: у стовпці рівня води відсутні 6,3% значень (6535 пропусків), у стовпці витрат води – 33,3% (34398 пропусків), а в обох стовпцях одночасно – 3,8% (3965 пропусків). Пропуски мають випадковий характер.

Для подальшого аналізу часові ряди перетворюються на вибірки випадкових величин. Планується розглянути рівень і витрати води за конкретний місяць року, агрегуючи дані для кожного окремого місяця без

прив'язки до дат. Аналіз виконується в межах кожного пункту спостереження, оскільки дані сильно залежать від географічного розташування. Наприклад, для аналізу даних за січень будуть вивчатися рівень і витрати води для всіх постів окремо. Заплановано проведення кореляційного та регресійного аналізу для виявлення залежностей між показниками [95]. При наявності значущих залежностей пропуски будуть заповнені за допомогою найбільш відповідної лінійної або квазілінійної моделі регресії.

Для виявлення взаємозв'язків між показниками рівня води та витрат води проведемо кореляційний і регресійний аналіз. Спочатку буде побудовано кореляційні поля для візуалізації зв'язків, що дозволить оцінити характер залежностей. Далі розрахуємо коефіцієнти кореляції Пірсона, Спірмена та Кендала для визначення лінійної або монотонної залежності. На основі отриманих результатів сформуємо відповідні регресійні моделі.

На рисунку 4.1 представлено приклади кореляційних полів для окремих постів [105], які дозволяють візуально оцінити наявність лінійного чи монотонного зв'язку. Додатково, у таблиці 4.1 наведено кількісну оцінку таких зв'язків, а також статистичну значущість коефіцієнтів кореляції.

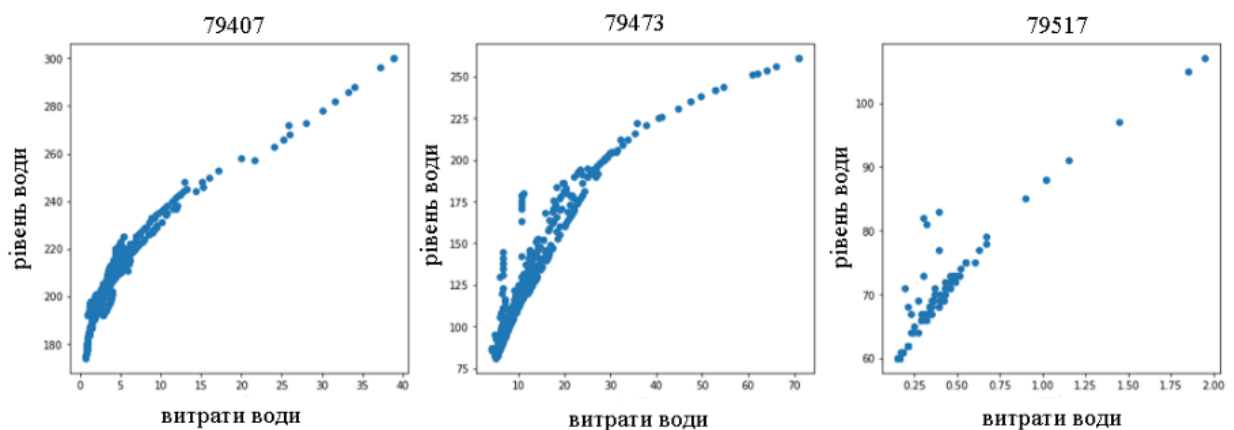


Рисунок 4.1 – Кореляційні поля для постів спостережень

79407, 79473, 79517

Коефіцієнти кореляції

Пост	Коефіцієнт кореляції	Оцінка	Значущий так / ні	Висновок
79043	Пірсона	0.877	так	лінійний
	Спірмена	0.873	так	монотонний
	Кендала	0.705	так	монотонний
79473	Пірсона	0.921	так	лінійний
	Спірмена	0.945	так	монотонний
	Кендала	0.847	так	монотонний
79477	Пірсона	0.896	так	лінійний
	Спірмена	0.902	так	монотонний
	Кендала	0.733	так	монотонний

Результати аналізу даних таблиці 4.1 підтверджують, що отримана інформація про характер вихідних даних створює підстави для застосування регресійного підходу до імпутування пропусків. Для цього будуть використані моделі лінійної регресії та 7 квазілінійних моделей, що можуть бути приведені до лінійної форми шляхом певних трансформацій даних.

Далі розглянемо результати навчання цих регресійних моделей на прикладі даних поста № 79473. У таблиці 4.2 наведено оцінки коефіцієнтів a та b , значення метрик R^2 і MSE , а також висновки щодо адекватності моделей. Серед усіх моделей за метрикою MSE (найменша середньо-квадратична похибка) найкращою виявилась модель № 3, тоді як за метрикою R^2 (найбільший коефіцієнт детермінації) лідером є модель № 4. Для наочної перевірки якості моделі побудовано кореляційне поле з регресійною лінією та довірчими інтервалами, які наведено на рисунку 4.2.

Побудовані моделі регресії

Модель	a	b	R^2	MSE	Адекватн.
1: $y = a + b * x$	-16.84	0.232	0.849	12.24	Так
2: $y = a + b * \ln(x)$	-136.2	30.93	0.759	19.53	Так
3: $y = \exp(a + b * x)$	0.6254	0.014	0.921	3.174	Так
4: $y = \exp(a + b * \ln(x))$	-6.982	1.945	0.923	8.393	Так

Продовження таблиці 4.2

Модель	a	b	R^2	MSE	Адекватн.
5: $y = \ln(a + b * x)$	-inf	inf	0.043	NaN	Ні
6: $y = a + b/x$	44.226	-3767.7	0.658	27.737	Так
7: $y = a + b * \exp(x)$	12.299	0.0	0.132	70.474	Так
8: $y = \sqrt{a + b * x}$	-1132.3	10.817	0.567	NaN	Так

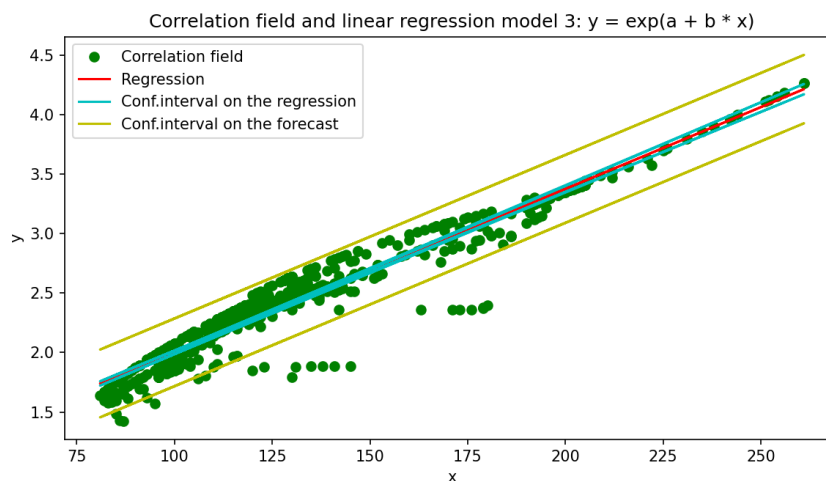


Рисунок 4.2 – Кореляційне поле та модель регресії № 3: $y = \exp(a + b * x)$

Збудовані регресійні моделі дозволяють отримати більш глибоке розуміння структури даних, що можна використовувати для заповнення пропусків. У випадках, коли один показник відсутній, але є інший, ці залежності допоможуть обчислювати відсутні значення з достатньою точністю.

Це дослідження було частково профінансоване Міністерством закордонних справ Чеської Республіки в межах проєкту № 24 РКVV UM 011 «Посилення стандартів викладання, досліджень та міжнародного співробітництва в Дніпровському національному університеті імені Олеся Гончара (ДНУ)», реалізованого Карловим університетом і ДНУ [121].

4.3 Аналіз точності імпутування та швидкодії (тест типу 1)

Для оцінки точності та швидкодії різних методів імпутування використовувалась частина датасету без пропусків, до якого штучно

– RegrValue – імплімент RegrImputer із спадним сортуванням ознак, з реакцією на патерн у даних та з корегуванням значення для категоріальної ознаки за найближчим значенням зі словника;

– RegrWeight – імплімент RegrImputer із спадним сортуванням ознак, з реакцією на патерн у даних та з корегуванням значення для категоріальної ознаки за найближчим за середньозваженою оцінкою значення зі словника;

– Entropy1/EntropyIter – імплімент EntropyImputer на основі ентропійного підходу, одно- та багатокроковий;

– EntropyIterThreads/Processes – імплімент EntropyImputer на основі ентропійного підходу, з використанням потоків або процесів;

– HybridRegrEntropy – гібридний імплімент HybridRegrEntropyImputer на основі ентропійного та регресійного підходів.

В стовпцях 10%, 20%, 30%, 40% наведено значення середньоквадратичного відхилення отриманих в результаті імпутування даних від вихідних даних (метрика RMSE) відповідно до обсягу пропусків. В стовпці Час наведений сумарний час роботи конкретного імпліменту для імпутування всіх варіантів обсягів пропусків. Курсивом відмічені стандартні імпліменти, з якими ми порівнюємо розроблені імпліменти.

Таблиця 4.3

Результати тесту типу 1 для датасету UCI, IgnoreNaNLabelEncoder

Метод	10%	20%	30%	40%	Загальний час, с
<i>SimpleImputer</i>	2.367	3.543	4.007	4.192	0.214
<i>IterativeImputer</i>	1.747	2.637	2.960	3.300	2.968
<i>kNNImputer</i>	1.859	2.633	3.063	3.436	0.696
<i>NoNa</i>	1.893	2.982	3.756	2.842	5.981
UnifiedCRSortedAsc	1.822	2.693	2.967	2.793	4.499
UnifiedCRSortedDesc	1.747	2.581	3.001	2.820	4.451
UnifiedCR2steps	1.727	2.593	3.027	2.753	7.922
RegrValue	1.711	2.580	3.017	2.743	0.643

Продовження таблиці 4.3

Метод	10%	20%	30%	40%	Загальний час, с
RegrWeight	1.717	2.580	3.020	2.750	0.646
Entropy1	6.947	8.723	10.583	10.517	8.186
EntropyIter	7.057	8.767	10.753	10.653	20.974
EntropyIterThreads	7.023	8.533	10.673	10.517	22.159
EntropyIterProcesses	6.843	8.647	10.740	10.560	15.944
HybridRegrEntropy	1.835	2.800	3.160	2.967	4.084

Жирним шрифтом на жовтому фоні у таблиці виділено ім'ютери, які показали найкращі результати за точністю або часом виконання. З аналізу таблиці 4.3 можна зробити наступні висновки:

1. Запропоновані ім'ютери `UnifiedClassRegrImputer` демонструють схожі результати за точністю та показують кращі результати порівняно зі своїм аналогом `NoNa` як за точністю, так і за швидкістю. Вони враховують патерни пропусків, реагуючи на кількість повних рядків чи наявність ознак із великою або малою кількістю пропусків.

2. Ім'ютер `RegrImputer` забезпечує вищу швидкість та частіше демонструє вищу точність імпутації.

3. У порівнянні з методом `IterativeImputer` із бібліотеки `scikit-learn`, запропоновані методи переважають, оскільки дозволяють декодувати результати для якісних ознак. Також слід зауважити, що в період активного розроблення власних методів та алгоритмів (2024 – 2025 роки) в бібліотеці `scikit-learn` клас `IterativeImputer` [96] помічений як експериментальний. Для його застосування в обчисленнях потрібно імпортувати маркерну ознаку `enable_iterative_imputer` з пакету `sklearn.experimental` як підтвердження розуміння користувача про те, що цей ім'ютер поки що не введений в експлуатацію.

4. Методи на основі ентропійного підходу поступаються іншим за точністю та швидкістю. Однак їхній гібридний варіант із використанням

регресії демонструє покращення точності імпутації та часу виконання, що робить його конкурентоздатним.

Візьмемо за базове значення (100%) результати методу NoNa за метрикою RMSE та швидкодією з таблиці 4.3. Для методу UnifiedClassRegrImputer виберемо найкращі показники з кожного стовпця таблиці, порівняємо їх із базовим значенням NoNa та обчислимо, на скільки відсотків покращився результат завдяки використанню саме UnifiedClassRegrImputer. Результати розрахунків представлено в таблиці 4.4.

Таблиця 4.4

Відсоткове покращення методу UnifiedClassRegrImputer відносно NoNa за метрикою RMSE та швидкодією для датасету UCI, IgnoreNaNLabelEncoder

Метод	10%	20%	30%	40%	Загальний час, с
NoNa	1.893	2.982	3.756	2.842	5.981
UnifiedClassRegrImputer	1.727	2.581	2.967	2.753	4.499
X	91.2%	86.6%	79.0%	96.9%	75.2%
100% – X	8.8%	13.4%	21.0%	3.1%	24.8%

В таблиці 4.4 жирним шрифтом на блакитному фоні представлені базові значення, отримані за методом NoNa, які ми приймаємо за 100%. В рядку X знаходяться значення, які показують, скільки склало значення метрики методу UnifiedClassRegrImputer відносно базового методу NoNa для кожного стовпця (наприклад, 91.2% означає, що результат UnifiedClassRegrImputer склав 91.2% від результату NoNa). В рядку (100% – X), який виділено жовтим кольором, можна побачити відсоток покращення методу UnifiedClassRegrImputer відносно NoNa. Наприклад, у стовпці, що відповідає вбудованим 10% пропусків, значення 8.8% означає, що метод UnifiedClassRegrImputer покращив результат на 8.8% порівняно з NoNa.

Таким чином, за точністю імпутування (метрика RMSE) метод UnifiedClassRegrImputer показав покращення від 3.1% до 21.0% залежно від

кількості пропусків даних (10%–40%). За швидкістю (загальний час, с) метод `UnifiedClassRegrImputer` виявився ефективнішим на 24.8%.

У таблицях 4.5 та 4.6 представлено результати для іншого способу перетворення якісних даних на кількісні за допомогою `IgnoreNaNFrequentEncoder(ascending=False)`. Таблиця 4.5 – результати для кардіологічного датасету UCI, в таблиці 4.6 – для датасету Framingham. В таблиці 4.5 в стовпці 30% наведено значення середньоквадратичного відхилення отриманих в результаті імпутування даних від вихідних даних (метрика RMSE) для обсягу пропусків 30%. В стовпці *Час* наведений час роботи заданого імп'ютера. Курсивом відмічені імп'ютери, з якими ми порівнюємо розроблені імп'ютери. Опис таблиці 4.6 аналогічний таблиці 4.3.

Таблиця 4.5

Результати тесту типу 1 для датасету UCI, `IgnoreNaNFrequentEncoder`

Метод	30%	Загальний час, с
<i>SimpleImputer</i>	3.81	0.009
<i>IterativeImputer</i>	3.01	0.197
<i>kNNImputer</i>	3.24	0.014
<i>NoNa</i>	3.17	1.176
<code>UnifiedCRSortedAsc</code>	3.07	1.103
<code>UnifiedCRSortedDesc</code>	3.07	1.122
<code>UnifiedCR2steps</code>	3.09	1.947
<code>RegrValue</code>	3.08	0.150
<code>RegrWeight</code>	3.08	0.147
<code>Entropy1</code>	6.05	3.691
<code>EntropyIter</code>	6.18	3.662
<code>EntropyIterThreads</code>	6.13	3.922
<code>EntropyIterProcesses</code>	6.17	4.920
<code>HybridRegrEntropy</code>	3.05	1.558

Результати тесту типу 1 для датасету Framingham, IgnoreNaNFrequentEncoder

Метод	10%	20%	30%	40%	Загальний час, с
<i>SimpleImputer</i>	2.01	3.68	2.75	4.51	0.048
<i>IterativeImputer</i>	1.63	3.13	2.38	4.03	2.934
<i>kNNImputer</i>	1.77	3.27	2.36	4.16	3.487
<i>NoNa</i>	1.89	3.32	2.42	4.38	6.238
UnifiedCR-SortedAsc	1.80	3.17	2.23	3.78	4.444
UnifiedCR-SortedDesc	1.73	3.09	2.41	4.03	4.573
UnifiedCR2steps	1.68	3.11	2.25	3.97	8.064
RegrValue	1.67	3.10	2.24	3.96	1.000
RegrWeight	1.67	3.09	2.24	3.96	1.018
Entropy1	11.70	16.20	15.60	21.62	17.760
EntropyIter	11.60	16.20	15.30	21.73	48.280
HybridRegrEntropy	1.77	3.29	2.31	4.11	4.975

Результати аналогічні попереднім: методи на основі регресії демонструють більшу швидкість, а точність імпутації залежить від патернів у даних та вибору методів. Лише метод на основі ентропійного підходу поступається за всіма показниками, хоча його покращений гібридний аналог демонструє результати, порівняні з іншими методами. Нижча точність та швидкодія ентропійного підходу для датасету Framingham пов'язані з більшою кількістю даних та більшою кількістю кількісних ознак порівняно з датасетом UCI.

Обчислимо, наскільки вдалося покращити точність імпутування на датасеті Framingham за рахунок застосування методу UnifiedClassRegrImputer порівняно з методом NoNa (позначимо його результати за 100%). Розрахунки наведено в таблиці 4.7. За точністю імпутування (метрика RMSE) метод UnifiedClassRegrImputer показав покращення від 6.9% до 13.7% залежно від кількості пропусків даних (10%–40%). За швидкістю (загальний час, с) метод UnifiedClassRegrImputer виявився ефективнішим на 28.8%.

Таблиця 4.7

Відсоткове покращення методу UnifiedClassRegrImputer відносно NoNa за метрикою RMSE та швидкодією для датасету UCI, IgnoreNaNLabelEncoder

Метод	10%	20%	30%	40%	Загальний час, с
NoNa	1.89	3.32	2.42	4.38	6.238
UnifiedClassRegrImputer	1.68	3.09	2.23	3.78	4.444
X	88.9%	93.1%	92.1%	86.3%	71.2%
100% – X	11.1%	6.9%	7.9%	13.7%	28.8%

В таблиці 4.8 наведено результати порівняння точності імпутування та швидкодії для датасету hydro_monitoring з гідрологічними даними. Цей датасет містить часові ряди. Аналогічно опису таблиці 4.3 в стовпцях 10%, 20%, 30%, 40% наведено значення середньоквадратичного відхилення отриманих в результаті імпутування даних від вихідних даних (метрика RMSE) відповідно до обсягу пропусків. В стовпці Час наведений сумарний час роботи конкретного імп'ютера для імпутування всіх варіантів обсягів пропусків. Курсивом відмічені імп'ютери, з якими ми порівнюємо розроблені імп'ютери. Останні 4 рядки таблиці стосуються результатів методу на основі виявленого кореляційного зв'язку. Corr – імп'ютер CorrelationImputer на основі виявленого кореляційного зв'язку між ознаками, виконаний послідовно; CorrThreads – імп'ютер CorrelationImputer, виконаний з використанням потоків; CorrProcesses – імп'ютер CorrelationImputer, виконаний з використанням процесів; CorrMonth – помісячне імпутування пропусків із застосуванням CorrelationImputer.

В таблиці 4.8 блакитним фоном позначено результати імпутування пропусків за методом CollerationImputer для всього датасету разом, а жовтим фоном позначено результати імпутування цього методу при його застосуванні для кожного місяця року окремо.

Результати тесту типу 1 для датасету hydro_monitoring

Метод	10%	20%	30%	40%	Загальний час, с
<i>SimpleFreq</i>	6.72	8.72	11.23	13.50	0.095
<i>SimpleMean</i>	5.59	7.37	9.51	11.22	0.071
<i>IterativeImputer</i>	2.62	2.58	3.32	4.27	0.347
<i>kNN</i>	1.74	2.07	2.80	3.12	18.310
UnifiedCRSortedAsc	3.36	3.61	4.60	5.86	0.235
UnifiedCRSortedDesc	5.08	6.65	8.70	10.25	0.235
UnifiedCR2steps	2.60	2.60	3.35	4.25	0.705
RegrValue	2.60	2.60	3.35	4.25	0.661
RegrWeight	2.60	2.60	3.35	4.25	0.641
Corr	1.56	1.88	2.41	2.83	1.572
CorrThreads	1.56	1.88	2.41	2.83	5.278
CorrProcesses	1.56	1.88	2.41	2.83	18.376
CorrMonth	1.21	1.51	1.80	2.15	15.176

З аналізу таблиці 4.8 можна зробити висновок, що метод *CollerationImputer*, який враховує кореляційні зв'язки, забезпечує кращу точність імпутації. Паралельне виконання цього методу зовсім не покращує швидкодію, бо витрачається більше часу на перемикання між потоками чи процесами, ніж на самі обчислення.

Таблиця 4.9

Відсоткове покращення методу *CorrelationImputerMonth* відносно простого *CorrelationImputer* для датасету hydro_monitoring

Метод	10%	20%	30%	40%	Загальний час, с
<i>CorrelationImputer</i>	1.56	1.88	2.41	2.83	1.572
<i>CorrelationImputerMonth</i>	1.21	1.51	1.8	2.15	15.176
X	77.6%	80.3%	74.7%	76.0%	965.4%
100% – X	22.4%	19.7%	25.3%	24.0%	-865.4%

В таблиці 4.9 наведено розрахунок відсоткового покращення методу імпутування `CorrelationImputer` в помісячному виконанні порівняно з простим методом `CorrelationImputer`, результати якого прийнято за 100%. Умовні позначення аналогічні таблицям 4.4, 4.7.

Метод помісячного імпутування дає найбільшу точність імпутування, що в середньому на 23% краще, ніж результати `CollerationImputer` для всього датасету разом, хоча він значно програє за швидкістю.

4.4 Аналіз впливу імпутування на точність моделі класифікації (тест типу 2)

Другий тип тестів спрямований на оцінку якості класифікації після застосування різних методів імпутації. Базовою моделлю, з якою будемо виконувати порівняння (base line в таблиці), є модель, яка навчається лише на повній частині датасету. Пропуски в датасетах імпутуються різними методами, в тому числі `SimpleImputer` з бібліотеки `scikit-learn Python` [89]. Після імпутування моделі бінарної класифікації навчаються на цих даних. Для оцінки якості моделі обчислюється метрика `accuracy` (точність класифікації) та `recall` (повнота) на тестовій вибірці. Щоб уникнути впливу випадковості, процедуру розбиття датасетів на навчальні та тестові вибірки повторюють кілька разів, усереднюючи результати метрик `accuracy` та `recall`.

Перед тренуванням моделей якісні ознаки перетворюються у кількісні. Потім виконуються імпутація пропусків і навчання моделей для задачі бінарної класифікації, використовуючи алгоритм `Random Forest` із підбором оптимальних гіперпараметрів.

У таблиці 4.10 наведено результати тестування для двох кардіологічних датасетів: `UCI Heart Disease Data` та `Framingham Heart Study`. Базові показники виділені жирним шрифтом на блакитному фоні, а найкращі результати за метриками `accuracy` або `recall` позначені жирним шрифтом на жовтому фоні.

Результати тесту типу 2 для оцінки якості моделі класифікації

Назва методу	Датасет UCI		Датасет Framingham	
	accuracy	recall	accuracy	recall
Base line	0.83	0.78	0.88	0.83
SimpleImputer	0.82	0.82	0.87	0.83
<i>IterativeImputer</i>	0.88	0.86	0.90	0.83
<i>kNNImputer</i>	0.84	0.85	0.90	0.84
<i>NoNa</i>	0.89	0.90	0.89	0.84
UnifiedCRSortedAsc	0.89	0.89	0.89	0.83
UnifiedCRSortedDesc	0.91	0.90	0.89	0.83
UnifiedCR2steps	0.91	0.90	0.89	0.83
EntropyIter	0.90	0.88	0.89	0.84
RegrValue	0.89	0.88	0.88	0.83
RegrWeight	0.91	0.90	0.90	0.85
HybridRegrEntropy	0.87	0.85	0.89	0.84

Основні висновки наступні:

1. Запропоновані методи імпутації суттєво покращують точність класифікації. Для датасету UCI Heart Disease Data показник accuracy підвищився з 83% до 91% (покращення на 9.6%), а recall – з 78% до 90% (покращення на 15.4%).

2. Для датасету Framingham Heart Study покращення менш значне, accuracy – на 2.3%, recall – на 2.4%, що пояснюється нижчим рівнем пропусків у початкових даних у порівнянні з UCI Heart Disease Data.

Якщо прийняти за 100% показники accuracy та recall, отримані на базових моделях, навчених лише на повній частині датасету, то можна розрахувати, на скільки відсотків змінюється якість класифікаційних моделей відносно базової. У таблиці 4.11 подано такі розрахунки: додатні значення у стовпцях ($x - 100\%$) вказують на те, на скільки відсотків покращилася якість

моделі за метрикою ассурасу. Базові значення (100%) виділено жирним шрифтом на блакитному фоні, а найкращі результати наведено на жовтому фоні.

Таблиця 4.11

Розрахунок зміни якості моделей класифікації за показниками ассурасу та recall після імпутування відносно базової моделі (у відсотках)

Метод	Датасет UCI			Метод	Датасет Framingham		
accuracy 0.83 = 100%	<i>x</i> %	<i>x-100%</i>		accuracy 0.88 = 100%	<i>x</i> %	<i>x-100%</i>	
HybridRegrEntropy	0.87	104.8	4.8	NoNa, UnifiedCRSortedAsc, EntropyIter, HybridRegrEntropy	0.89	101.1	1.1
NoNa, UnifiedCRSortedAsc, RegrValue	0.89	107.2	7.2	RegrWeight	0.90	102.3	2.3
EntropyIter	0.90	108.4	8.4				
UnifiedCRSortedDesc, UnifiedCR2steps, RegrWeight	0.91	109.6	9.6				
recall 0.78 = 100%	<i>x</i> %	<i>x-100%</i>		recall 0.83 = 100%	<i>x</i> %	<i>x-100%</i>	
HybridRegrEntropy	0.85	109.0	9.0	NoNa, EntropyIter, HybridRegrEntropy	0.84	101.2	1.2
EntropyIter, RegrValue	0.88	112.8	12.8	RegrWeight	0.85	102.4	2.4
UnifiedCRSortedAsc	0.89	114.1	14.1				
NoNa, UnifiedCRSortedDesc, UnifiedCR2steps, RegrWeight	0.90	115.4	15.4				

Таким чином, застосування ефективних методів імпутування сприяє покращенню якості моделей класифікації, особливо у випадках із значною кількістю пропусків у даних, як у датасеті UCI.

4.5 Аналіз впливу імпутування на точність моделі прогнозування (тест типу 3)

Пропуски у часових рядах не можна ігнорувати з кількох причин:

1. Втрата інформації. Ігнорування пропусків веде до втрати даних, що може зробити аналіз необ'єктивним. Втрачені значення можуть містити інформацію про поведінку ряду в певні моменти часу або мати певну структуру.

2. Порушення часових залежностей. Значення у часових рядах часто залежать одне від одного. Пропуски розривають ці зв'язки, ускладнюючи побудову моделей та прогнозування. Наприклад, пропущені точки можуть спотворювати зв'язок між значеннями.

3. Викривлення статистичних властивостей. Пропуски можуть змінювати середнє, дисперсію та інші характеристики даних, що погіршує результати моделювання і прогнозування.

4. Проблеми з моделями. Моделі часових рядів, таких як ARIMA чи експоненційне згладжування, потребують повноти даних із рівномірними інтервалами часу. Ігнорування пропусків може зробити моделі непридатними для використання або знизити точність результатів.

5. Вплив на тренди і сезонність. Пропуски впливають на виявлення тенденцій і сезонних змін, що може приховати реальні зміни або створити хибну ілюзію тренду.

6. Неправильна інтерпретація пропусків. Випадковість пропусків не завжди очевидна: вони можуть сигналізувати про аномалії чи зміни в системі. Їх ігнорування позбавляє важливої інформації.

7. Ускладнення прогнозування. Прогнозування за неповними даними менш точне, особливо якщо пропуски є у важливих точках, таких як піки або спади.

8. Неправильна оцінка ризиків. У застосуваннях, де важливе моделювання ризиків, ігнорування пропусків може призвести до невірних управлінських рішень.

Таким чином, ігнорування пропусків у часових рядах може привести до суттєвих помилок в аналізі, оцінці параметрів і прогнозах, що є неприйнятним для більшості реальних задач. Кращім рішенням буде використовувати методи

імпутування, які дозволять заповнити пропуски, керуючись визначеною логікою та мінімізувати негативний вплив пропусків на результати аналізу.

У цьому дослідженні будемо порівнювати вплив на якість моделей прогнозування імпутування пропусків за допомогою розроблених методів, в тому числі звичайним алгоритмом *CorrelationImputer* (алгоритм 2.7) та таким, який виконується для кожного місяця окремо (алгоритм 2.8). При імпутуванні для кожного місяця окремо будемо перетворювати часові ряди на випадкові величини, такі як рівень та витрати води за конкретний місяць року. Слід зауважити, що імпутування виконуємо за кожним окремим постом спостережень, оскільки дані мають сильну залежність від географічного розташування. З набору даних відбираємо інформацію про витрати і рівень води за обраний місяць усіх років спостережень для заданого посту, що формує сукупні дані для одного місяця без прив'язки до дат. Виконуємо підбір кращої регресійної залежності, а потім, використовуючи цю залежність, виконуємо крок імпутування однієї ознаки за даними іншої. Процедура повторюємо для кожного місяця окремо. Далі знову переходимо до часових рядів.

Будемо здійснювати прогнозування методами *XGBoost* [97, 98], *Prophet* [99–101], *ARIMA* [102–104].

Спочатку виконаємо імпутування пропусків у даних різними методами. Потім розділимо кожний часовий ряд на навчальний та тестовий набори. Виконаємо прогнозування на 15 днів. Для порівняння прогнозу з реальними даними візьмемо метрику середньо-квадратичного відхилення (*RMSE*). У даних присутня сезонність з довжиною періоду, що дорівнює 365 днів (1 рік). В таблиці 4.12 представлено результати прогнозування витрат води на 15 днів.

Жирним шрифтом на блакитному фоні показано результати базових моделей, що отримані за наборами даних, з яких видалено пропуски. Жирним шрифтом на жовтому фоні позначено найкращі результати у порівнянні з базовим результатом. З аналізу таблиці можна побачити, що майже завжди якість моделей прогнозування покращується при виконанні імпутування пропущених даних. *SimpleImputer* може показувати погіршення моделей.

Найкращі результати показав імпультер на основі кореляційного зв'язку (особливо той, що виконував помісячне імпутовання пропусків) та імпультер на основі класифікатору та регресору, що в цілому прогнозовано, бо ці два імпультери здатні враховувати кореляційний зв'язок між ознаками.

Таблиця 4.12

Результати прогнозування

Метод	XGBoost	Prophet	ARIMA
Base line	0.7837	4.1529	1.912
SimpleImputer	0.6956	4.6856	1.2030
<i>IterativeImputer</i>	0.5448	3.1888	1.0756
<i>kNN</i>	0.5419	3.4968	1.1191
Corr	0.5377	3.5162	0.3674
CorrMonth	0.5136	3.0550	0.4701
UnifiedCRSortedAsc	0.5448	3.1519	1.0616
UnifiedCRSortedDesc	0.5448	3.1519	1.0616
UnifiedCR2steps	0.5448	3.1519	1.0616
RegrValue	0.5448	3.1519	1.0616
RegrWeight	0.5448	3.1519	1.0616

Якщо брати за 100% значення RMSE, отримані на базових моделях з ігноруванням пропусків у даних, то можна обчислити у відсотках, наскільки змінюється якість моделей прогнозування відносно базової моделі. В таблиці 4.13 наведені розрахунки, де додатне значення у стовпцях ($100\% - x$) показує, на скільки відсотків якість моделі прогнозування покращується за показником RMSE, а від'ємне – відповідно відсоток погіршення. Жирним шрифтом на блакитному фоні позначено базові значення, які прийнято за 100%. Жовтим фоном позначено найкращі показники.

Розрахунок зміни якості моделей прогнозування за показником RMSE після імпутування відносно базової моделі (у відсотках)

Метод	XGBoost			Prophet			ARIMA		
Base line	0.7837 = 100%			4.1529 = 100%			1.912 = 100%		
		<i>x%</i>	<i>100% - x</i>		<i>x%</i>	<i>100% - x</i>		<i>x%</i>	<i>100% - x</i>
<i>Simple</i>	0.6106	77.9	22.1	4.4142	106.3	-6.3	1.1986	62.7	37.3
Corr	0.5377	68.6	31.4	3.5162	84.7	15.3	0.3674	19.2	80.8
CorrM	0.5136	65.5	34.5	3.055	73.6	26.4	0.4701	24.6	75.4
UnifiedCR	0.5448	69.5	30.5	3.1519	75.9	24.1	1.0616	55.5	44.5

Так модель прогнозування XGBoost покращено на 34.5%, Prophet – на 26.4% (обидва результати після імпутування методом CorrelationImputer з підбором моделей для кожного місяця року окремо), ARIMA – на 80.8% (метод імпутування CorrelationImputer). В той же час SimpleImputer для метода Prophet навіть погіршив модель на 6.3%.

4.6 Аналіз змін ентропії до та після імпутування (тест типу 4)

У цьому тесті розраховується умовна ентропія за формулою (2.7) для кожного методу. Для кожного методу виконується імпутування ітераційно, поки умовна ентропія зменшується для поточного методу. Уточнення значень інтерполяції проводиться по кожній позиції з пропуском, вважаючи, що інші імпутовані значення вже інтегровані в датасет. Далі результати умовної ентропії порівнюються з початковими значеннями. Як приклад, використовувався датасет UCI, де якісні порядкові ознаки були перетворені на кількісні за допомогою методу IgnoreNaNLabelEncoder.

В таблиці 4.14 наведено результати аналізу умовної ентропії до та після імпутування. Для демонстрації було вибрано кілька різних типів ознак:

- chol: рівень холестерину (неперервна величина);

– restecg: результат електрокардіографії у стані спокою (якісна порядкова величина);

– sex: стать (бінарна величина).

В таблиці 4.14 для набору даних з 30% штучно внесених пропусків:

– жирним шрифтом на блакитному фоні виділено базовий рядок – значення умовної ентропії до імпутування;

– жирним шрифтом на жовтому фоні виділено комірки з найменшими значеннями умовної ентропії (на основі ентропії EntropyImputer);

– на рожевому фоні – комірки з найбільшими значеннями умовної ентропії (SimpleImputer);

– стрілочками вгору або вниз позначено приклади збільшення чи зменшення ентропії відносно базового значення.

Таблиця 4.14

Зміна значення умовної ентропії після імпутування

Метод	chol	restecg	sex
Base line	0.790399	0.969341	0.963700
<i>SimpleImputer</i>	0.790474 ↑	0.979026 ↑	0.968079 ↑
<i>IterativeImputer</i>	0.780214	0.908985	0.872935 ↓
<i>kNNImputer</i>	0.778758	0.889923 ↓	0.899879 ↓
UnifiedCRSortedAsc	0.779789	0.960247	0.965925
UnifiedCRSortedDesc	0.777792	0.946205	0.951608
UnifiedCR2steps	0.779834	0.968507	0.962079
RegrValue	0.779374	0.969216	0.957166
RegrWeight	0.779834	0.968042	0.964316
Entropy1	0.624774 ↓	0.86325 ↓	0.858682 ↓
EntropyIter	0.622365 ↓	0.86325 ↓	0.858682 ↓
HybridRegrEntropy	0.779374	0.86325 ↓	0.958682

З аналізу таблиці 4.14 видно, що більшість методів імпутування сприяють зменшенню базової ентропії. Найкращий результат демонструє

метод EntropyImputer, заснований на ентропійному підході. Більше зменшення умовної ентропії спостерігається для неперервних величин, тоді як для бінарних величин зменшення ентропії є найменш значним.

У таблиці 4.15 показано поетапне зменшення умовної ентропії на кожній ітерації алгоритму ентропійного методу в його ітераційному виконанні. З неї видно, що вже після 4-ї ітерації значення ентропії перестають змінюватися для всіх розглянутих ознак. Для демонстрації було вибрано кілька різних типів ознак:

- **trestbps**: артеріальний тиск у стані спокою (неперервна величина);
- **chol**: рівень холестерину (неперервна величина);
- **thalach**: максимальна частота серцевих скорочень (неперервна величина);
- **ca**: кількість основних судин (0-3), забарвлених за допомогою флюороскопії (якісна порядкова величина).

Таблиця 4.15

Кроки зменшення умовної ентропії на кожній ітерації алгоритму

Номер ітерації	trestbps	chol	thalach	ca
0	0.6140	0.683	0.526	0.779
1	0.5764	0.654	0.494	0.709
2	0.5760	-	0.492	0.708
3	-	-	-	0.706
4	-	-	-	0.705
5	-	-	-	-

Далі всі методи імпутування застосовувалися ітераційно, доки значення умовної ентропії продовжувало зменшуватися (за принципом, аналогічним до роботи ітераційного ентропійного методу). Процес зупинявся, коли зменшення ентропії припинялося або досягалося заздалегідь задане максимальне число ітерацій.

Для ілюстрації підходу у таблиці 4.16 наведено результати імпутування за окремими ознаками датасету UCI. Перед обробкою якісні ознаки були перетворені на кількісні за допомогою методу IgnoreNaNLabelEncoder.

Таблиця 4.16

Зміна значення умовної ентропії
після застосування методів імпутування ітераційно

Метод	chol	restecg	sex
Base line	0.791235	0.970871	0.967796
	0.790242 ↓	0.972011 ↑	0.965850 ↓
SimpleImputer	0.796228 ↑	0.975102 ↑	0.967657 ↑
	0.793609	0.972962	0.970045
<i>IterativeImputer</i>	0.791185	0.938441	0.949335
	0.780171	0.959821	0.830418
<i>kNNImputer</i>	0.789338	0.922918	0.949519
	0.783518	0.959821	0.820332
UnifiedCRSortedAsc	0.787801	0.964656	0.967657
	0.77828	0.971386	0.970045
UnifiedCRSortedDesc	0.795379	0.964656	0.967657
	0.782818	0.971386	0.968787
UnifiedCR2steps	0.79057	0.965685	0.967657
	0.778978	0.971386	0.969433
EntropyIter	0.622744 ↓	0.875675 ↓	0.960461 ↓
	0.613428 ↓	0.961728 ↑	0.976096 ↑

Так само, як зазначено у висновках до таблиці 4.14, дані таблиці 4.16 свідчать про те, що метод SimpleImputer не сприяє зменшенню ентропії, тоді як інші методи зменшують її, і найкращим залишається EntropyImputer. Проте уточнення значень на додаткових ітераціях майже не впливає на результат, а іноді навіть погіршує умовну ентропію вже на другій ітерації.

4.7 Демонстрація роботи методів у конвеєрі (pipeline)

Запропоновані методи імпутації пропусків розроблено відповідно до архітектурних принципів бібліотеки scikit-learn, що забезпечує їхню сумісність

з конвеєрами pipeline. Як ілюстрацію наведено побудову класифікаційного конвеєра з використанням одного з розроблених імпультерів (лістинг 4.1).

Лістинг 4.1 – Код для запуску конвеєру

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score,
confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from zo_ignore_nan_encoders import IgnoreNaNLabelEncoder
from zo_imputers import RegrImputer
from zo_test_imputers.utils import get_ready_na_UCI

def pipeline_RF():
    data = get_ready_na_UCI(dropna=False)
    encoder = IgnoreNaNLabelEncoder()
    data_1 = encoder.fit_transform(data)
    target = 'num'
    # Створення pipeline із заповненням пропусків,
    # масштабуванням та класифікацією
    pipeline = Pipeline([
        ('imputer', RegrImputer(measure='value')),
        ('scaler', MinMaxScaler()),
        ('classifier', RandomForestClassifier(n_estimators=100))
    ])
    # Розподіл даних на X і y
    X = data_1.drop(target, axis=1)
    y = data_1[target]
    # Розподіл на навчальний та тестовий набори
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=0.3, random_state=42)
    # Навчання і передбачення
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    conf_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred),
                               index=['actual 0', 'actual 1'],
                               columns=
                               ['predicted 0', 'predicted 1'])
    print("Confusion matrix:")
    print(conf_matrix)
    print(f'Accuracy on Test Set: {accuracy:.2f}')
    print(f'Recall score: {recall:.2f}')

if __name__ == "__main__":
    pipeline_RF()
```

Опис роботи коду:

1. Завантажуємо датасет UCI, у якому пропуски залишаються для обробки.
2. Використовуємо IgnoreNaNLabelEncoder для перетворення якісних ознак на кількісні.
3. Будуємо конвеєр із трьома етапами:
 - RegrImputer: метод імпутації, що базується на регресійному підході;
 - MinMaxScaler: масштабування даних;
 - RandomForestClassifier: класифікатор із 100 деревами.
4. Розділяємо дані на навчальну та тестову вибірки (70% і 30% відповідно).
5. Навчаємо конвеєр на тренувальних даних і прогнозуємо значення для тестового набору.
6. Оцінюємо точність моделі (accuracy) та метрику повноти (recall), а також будуємо матрицю помилок (confusion matrix).

Результат виконання цього коду зображений на рисунку 4.5.

```
Confusion matrix:
              predicted 0  predicted 1
actual 0           99      21
actual 1           27     129
Accuracy on Test Set: 0.83
Recall score: 0.83
```

Рисунок 4.5 – Результат запуску тесту у конвеєрі

Таким чином, продемонстровано, що запропоновані ім'ютери успішно інтегруються в архітектуру scikit-learn, відповідають вимогам сумісності з pipeline та забезпечують гнучкість при вирішенні задач класифікації з відновленням пропусків.

4.8 Висновок до розділу 4

Цей розділ містить аналіз результатів роботи [120].

1. Оцінено ефективність нових методів за точністю імпутування та часом обробки на датасетах медичного моніторингу:

1.1. Метод `UnifiedClassRegrImputer` демонструє більшу ефективність, ніж його аналог метод `NoNa`, як за точністю, так і за швидкодією:

– на датасеті `UCI` точність покращується від 3% до 21% залежно від кількості пропусків, швидкодія – до 24%;

– на датасеті `Framingham` точність покращується від 7% до 14% залежно від кількості пропусків, швидкодія – до 29%;

– за різними налаштуваннями він здатний враховувати патерни у даних, такі як наявність певної кількості повних рядків даних та певної кількості рядків лише з одним пропуском у рядку, а також реагувати скоріше на ознаки з більшою кількістю пропусків або з меншою.

1.2. Метод `RegrImputer` демонструє найбільшу ефективність за часом виконання та в більшості випадків – за точністю імпутування.

1.3. Метод `EntropyImputer` демонструє найменшу ефективність за точністю імпутування (метрика `RMSE`) та часом виконання, і найбільшу – за показником зменшення умовної ентропії ознак.

1.4. Гібридний метод `HybridRegrEntropyImputer` дозволяє покращити показники точності імпутування та значно скоротити час обчислень відносно `EntropyImputer`, і за показником точності імпутування демонструє схожі результати з `UnifiedClassRegrImputer` та `RegrImputer`.

2. Оцінено ефективність нових методів за точністю імпутування та часом обробки на датасеті гідрогеологічного моніторингу:

2.1. Метод `CorrelationImputer` при наявності кореляційного зв'язку між ознаками забезпечує найкращу точність імпутації; час виконання цього методу у послідовному застосуванні є схожим з іншими методами.

2.2. Для часових рядів з сезонною складовою метод імпутування з підбором моделей для кожного місяця року дає більшу точність імпутування у порівнянні зі своїм простим еквівалентом `CorrelationImputer` в середньому на 23%, але виконується набагато повільніше.

3. Проаналізовано вплив імпутування на якість моделі класифікації для датасету `UCI Heart Disease Data`. Аналіз показав, що всі розглянуті методи покращують якість за показниками `accuracy` та `recall` у порівнянні з базовими моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером `SimpleImputer`.

3.1. Показник `accuracy` покращився з 83% до 91%, зокрема при імпутуванні `HybridRegrEntropyImputer` – на 4.8% , `EntropyImputer` – на 8.4%, `UnifiedClassRegrImputer`, `RegrImputer` – на 9.6%.

3.2. Показник `recall` покращився з 78% до 90%, зокрема при імпутуванні `HybridRegrEntropyImputer` – на 9% , `EntropyImputer` – на 12.8%, `UnifiedClassRegrImputer`, `RegrImputer` – на 15.4%.

4. Проаналізовано вплив імпутування на якість моделі класифікації для датасету `Framingham Heart Study`. Аналіз показав, що всі розглянуті методи покращують якість моделі за показниками `accuracy` та `recall` у порівнянні з базовими моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером `SimpleImputer`.

4.1. Показник `accuracy` покращився з 88% до 90%, зокрема при імпутуванні `HybridRegrEntropyImputer`, `EntropyImputer`, `UnifiedClassRegrImputer` – на 1.1%, `RegrImputer` – на 2.3%.

4.2. Показник `recall` покращився з 83% до 85%, зокрема при імпутуванні `HybridRegrEntropyImputer`, `EntropyImputer`, `UnifiedClassRegrImputer` – на 1.2%, `RegrImputer` – на 2.4%.

5. Проаналізовано вплив імпутування на якість моделей прогнозування для датасету `hydro_monitoring`. Аналіз показав, що всі розглянуті методи покращують точність прогнозу за метрикою `RMSE` у порівнянні з базовими

моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером SimpleImputer.

5.1. Методи імпутування UnifiedClassRegrImputer, RegrImputer та HybridRegrImputer демонструють однаковий результат, оскільки в датасеті немає якісних ознак, і тому працює один і той самий алгоритм з використанням регресору.

5.2. Модель XGBoost покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 30.5%, CorrelationImputer – на 31.4%, CorrelationImputer з підбором моделі для кожного місяця року – на 34.5%.

5.3. Модель Prophet покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 24.1%, CorrelationImputer – на 15.3%, CorrelationImputer з підбором моделі для кожного місяця року – на 26.4%.

5.4. Модель ARIMA покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 44.5%, CorrelationImputer – на 80.8%, CorrelationImputer з підбором моделі для кожного місяця року – на 75.4%.

6. Продемонстровано, що запропоновані імп'ютери інтегруються в екосистему scikit-learn та відповідають вимогам сумісності з pipeline.

ВИСНОВКИ

1. В роботі проведено аналіз сучасного стану досліджень у галузі імпутації пропусків у даних. Визначено, що однією з ключових проблем у попередньому аналізі є виявлення пропущених значень. Найбільша складність полягає у відсутності універсального алгоритму, який міг би ефективно працювати для всіх типів задач. Для кожного конкретного випадку необхідно підбирати найбільш підходящі відомі методи, їх комбінації, модифікації або навіть розробляти нові підходи.

2. Розроблено та досліджено методи імпутування пропусків у даних:

2.1. Метод `UnifiedClassRegrImputer`, оснований на застосуванні класифікатору для якісних ознак та регресору – для кількісних, який є удосконаленням методу `NoNa`.

2.2. Метод `RegrImputer` на основі застосування регресору з корегуванням значення для якісних ознак.

2.3. Метод `EntropyImputer`, що є удосконаленням методу на основі ентропійного підходу.

2.4. Гібридний метод `HybridRegrEntropyImputer`, який поєднує ентропійний та регресійний підходи.

2.5. Метод `CorrelationImputer` на основі виявленого кореляційного зв'язку між ознаками.

3. Удосконалено методи перетворення якісних ознак на кількісні шляхом збереження інформації про пропуски та можливістю виконувати зворотне перетворення:

– `IgnoreNaNLabelEncoder` – модифікація стандартного `LabelEncoder`, яка не кодує пропуски, зберігає словник та дозволяє зворотне перетворення;

– `IgnoreNaNFrequentEncoder` – модифікація `LabelEncoder` та `FrequencyEncoder`, яка враховує частоту значень, надаючи більшу вагу найпоширенішим або найрідшим категоріям, не кодує пропуски, зберігає словник та дозволяє зворотне перетворення.

4. Всі методи реалізовано у вигляді класів на мові програмування Python відповідно до архітектурних принципів бібліотеки scikit-learn, що забезпечує уніфіковане застосування та сумісність з pipeline.

5. Оцінено ефективність нових методів за точністю імпутування та часом обробки на датасетах медичного моніторингу:

5.1. Метод UnifiedClassRegrImputer демонструє більшу ефективність, ніж його аналог метод NoNa, як за точністю, так і за швидкістю:

– на датасеті UCI точність покращується від 3% до 21% залежно від кількості пропусків, швидкість – до 24%;

– на датасеті Framingham точність покращується від 7% до 14% залежно від кількості пропусків, швидкість – до 29%;

– за різними налаштуваннями він здатний враховувати патерни у даних, такі як наявність певної кількості повних рядків даних та певної кількості рядків лише з одним пропуском у рядку, а також реагувати скоріше на ознаки з більшою кількістю пропусків або з меншою.

5.2. Метод RegrImputer демонструє найбільшу ефективність за часом виконання та в більшості випадків – за точністю імпутування.

5.3. Метод EntropyImputer демонструє найменшу ефективність за точністю імпутування (метрика RMSE) та часом виконання, і найбільшу – за показником зменшення умовної ентропії ознак.

5.4. Гібридний метод HybridRegrEntropyImputer дозволяє покращити показники точності імпутування та значно скоротити час обчислень відносно EntropyImputer, і за показником точності імпутування демонструє схожі результати з UnifiedClassRegrImputer та RegrImputer.

6. Оцінено ефективність нових методів за точністю імпутування та часом обробки на датасеті гідрогеологічного моніторингу:

6.1. Метод CorrelationImputer при наявності кореляційного зв'язку між ознаками забезпечує найкращу точність імпутації; час виконання цього методу у послідовному застосуванні є схожим з іншими методами.

6.2. Для часових рядів з сезонною складовою метод імпутування з підбором моделей для кожного місяця року дає більшу точність імпутування у порівнянні зі своїм простим еквівалентом `CorrelationImputer` в середньому на 23%, але виконується набагато повільніше.

7. Проаналізовано вплив імпутування на якість моделі класифікації для датасету `UCI Heart Disease Data`. Аналіз показав, що всі розглянуті методи покращують якість за показниками `accuracy` та `recall` у порівнянні з базовими моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером `SimpleImputer`.

7.1. Показник `accuracy` покращився з 83% до 91%, зокрема при імпутуванні `HybridRegrEntropyImputer` – на 4.8% , `EntropyImputer` – на 8.4%, `UnifiedClassRegrImputer`, `RegrImputer` – на 9.6%.

7.2. Показник `recall` покращився з 78% до 90%, зокрема при імпутуванні `HybridRegrEntropyImputer` – на 9% , `EntropyImputer` – на 12.8%, `UnifiedClassRegrImputer`, `RegrImputer` – на 15.4%.

8. Проаналізовано вплив імпутування на якість моделі класифікації для датасету `Framingham Heart Study`. Аналіз показав, що всі розглянуті методи покращують якість моделі за показниками `accuracy` та `recall` у порівнянні з базовими моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером `SimpleImputer`.

8.1. Показник `accuracy` покращився з 88% до 90%, зокрема при імпутуванні `HybridRegrEntropyImputer`, `EntropyImputer`, `UnifiedClassRegrImputer` – на 1.1%, `RegrImputer` – на 2.3%.

8.2. Показник `recall` покращився з 83% до 85%, зокрема при імпутуванні `HybridRegrEntropyImputer`, `EntropyImputer`, `UnifiedClassRegrImputer` – на 1.2%, `RegrImputer` – на 2.4%.

9. Проаналізовано вплив імпутування на якість моделей прогнозування для датасету `hydro_monitoring`. Аналіз показав, що всі розглянуті методи покращують точність прогнозу за метрикою `RMSE` у порівнянні з базовими

моделями, для яких пропущені значення відкинуті, або імпутування виконано найпростішим імп'ютером SimpleImputer.

9.1. Методи імпутування UnifiedClassRegrImputer, RegrImputer та HybridRegrImputer демонструють однаковий результат, оскільки в датасеті немає якісних ознак, і тому працює один і той самий алгоритм з використанням регресору.

9.2. Модель XGBoost покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 30.5%, CorrelationImputer – на 31.4%, CorrelationImputer з підбором моделі для кожного місяця року – на 34.5%.

9.3. Модель Prophet покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 24.1%, CorrelationImputer – на 15.3%, CorrelationImputer з підбором моделі для кожного місяця року – на 26.4%.

9.4. Модель ARIMA покращилась при імпутуванні HybridRegrEntropyImputer, RegrImputer, UnifiedClassRegrImputer – на 44.5%, CorrelationImputer – на 80.8%, CorrelationImputer з підбором моделі для кожного місяця року – на 75.4%.

10. Результати дисертаційних досліджень впроваджено в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара. Розроблені методи та програмне забезпечення використовуються при викладанні дисциплін «Аналіз даних на мові Python», «Оптимізація та підвищення продуктивності програмного коду», «Аналіз та візуалізація даних», «Технології пошуку структури в даних» для здобувачів першого (бакалаврського) рівнів вищої освіти галузі інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Preparing Your Dataset for Machine Learning: 10 Basic Techniques That Make Your Data Better. *Alexsoft Software R&D Engineering*. 2021. URL: <https://www.altexsoft.com/blog/datascience/preparing-your-dataset-for-machine-learning-8-basic-techniques-that-make-your-data-better/>.
2. Kutiname S., Millham R., Adekoya A. F., Tettey M., Weyori B. A., Appiahene P. Application of Machine Learning Algorithms in Coronary Heart Disease: A Systematic Literature Review and Meta-Analysis. *International Journal of Advanced Computer Science and Applications (IJACSA)*. Vol. 13, No. 6. 2022. DOI: <http://dx.doi.org/10.14569/IJACSA.2022.0130620>.
3. Yahaya L., Oye N., Garba E. A Comprehensive Review on Heart Disease Prediction Using Data Mining and Machine Learning Techniques. *American Journal of Artificial Intelligence*. Vol. 4. P. 20–29. 2020. DOI: 10.11648/j.ajai.20200401.12.
4. Hasan M. A. M., Shin J., Das U., Srizon A. Y. Identifying Prognostic Features for Predicting Heart Failure by Using Machine Learning Algorithm. *Proceedings of the 2021 11th International Conference on Biomedical Engineering and Technology*. P. 40–46, 2021. DOI: <https://doi.org/10.1145/3460238.3460245>.
5. Tardif J. C. Coronary artery disease in 2010. *European Heart Journal, Supplements*. Vol. 12, No. SUPPL. C. P. 2–10. 2010. DOI: <https://doi.org/10.1093/eurheartj/suq014>.
6. Khan M. A. et al. Global Epidemiology of Ischemic Heart Disease: Results from the Global Burden of Disease Study. *Cureus*. Vol. 12, No. 7. 2020. DOI: <https://doi.org/10.7759/cureus.9349>.
7. N. Kumar and D. Kumar. Machine Learning based Heart Disease Diagnosis using Non-Invasive Methods: A Review. *Journal of Physics: Conference Series*. Vol. 1950, No. 1. 2021. DOI: <https://doi.org/10.1088/1742-6596/1950/1/012081>.
8. Haleem A., Javaid M., Singh R. P., Suman R. Applications of Artificial Intelligence (AI) for cardiology during COVID-19 pandemic. *Sustainability in*

Operations and Computing. Vol. 2, No. February. P. 71–78. 2021. DOI: <https://doi.org/10.1016/j.susoc.2021.04.003>.

9. Johnson K. W. et al. Artificial Intelligence in Cardiology. *Journal of the American College of Cardiology*. Vol. 71, No. 23. P. 2668–2679. 2018. DOI: <https://doi.org/10.1016/j.jacc.2018.03.521>.

10. Constantinides P., Fitzmaurice D. A. Artificial intelligence in cardiology: Applications, benefits and challenges. *British Journal of Cardiology*. Vol. 25, No. 3. P. 86–87. 2018. DOI: <https://doi.org/10.5837/bjc.2018.024>.

11. Talabis M. R. M., McPherson R., Miyamoto I., Martin J. L., Kaye D. Analytics Defined. *Information Security Analytics*. P. 1–12. 2015. DOI: <https://doi.org/10.1016/B978-0-12-800207-0.00001-0>.

12. Sardar P., Abbott J. D., Kundu A., Aronow H. D., Granada J. F., Giri J. Impact of Artificial Intelligence on Interventional Cardiology: From Decision-Making Aid to Advanced Interventional Procedure Assistance. *JACC: Cardiovascular Interventions*. Vol. 12, No. 14. P. 1293–1303. 2019. DOI: <https://doi.org/10.1016/j.jcin.2019.04.048>.

13. Janosi A., Steinbrunn W., Pfisterer M., Detrano R. Heart Disease [Электронный ресурс]. *UCI Machine Learning Repository*. 1988. DOI: <https://doi.org/10.24432/C52P4X>.

14. Heart Disease Data Set from UCI data repository [Электронный ресурс]. *Kaggle*. URL: <https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data>.

15. Framingham Heart Study-Cohort (FHS-Cohort) [Электронный ресурс]. *NHLBI*. URL: <https://biolincc.nhlbi.nih.gov/studies/framcohort/>.

16. Framingham heart study dataset [Электронный ресурс]. *Kaggle*. URL: <https://www.kaggle.com/datasets/aasheesh200/framingham-heart-study-dataset>.

17. Rani P., Kumar R., Sid N. M. O., Anurag A. A decision support system for heart disease prediction based upon machine learning. *Journal of Reliable Intelligent Environments*. Vol. 7. P.263–275. 2021. DOI: <https://doi.org/10.1007/s40860-021-00133-6>.

18. Renugadevi G., Priya G. A., Sankari B. D., Gowthamani R. Predicting heart disease using hybrid machine learning model. *Journal of Physics: Conference Series*. Vol. 1916, No. 1. 2021. DOI: <https://doi.org/10.1088/1742-6596/1916/1/012208>.
19. Yuvali M., Yaman B., Tosun Ö. Classification Comparison of Machine Learning Algorithms Using Two Independent CAD Datasets. *Mathematics*. Vol. 10, P. 311. 2022. DOI: <https://doi.org/10.3390/math10030311>.
20. Kibria H. B., Matin A. The severity prediction of the binary and multi-class cardiovascular disease – A machine learning-based fusion approach. *Computational Biology and Chemistry*. Vol. 98, P.107672. 2022. ISSN 1476-9271. DOI: <https://doi.org/10.1016/j.compbiolchem.2022.107672>.
21. Alizadehsani R., Habibi J., Sani Z., Mashayekhi H., Boghrati R., Ghandeharioun A., Bahadorian B. Diagnosis of Coronary Artery Disease Using Data Mining Based on Lab Data and Echo Features. *Journal of Medical and Bioengineering*. Vol. 1. P. 26-29. 2012. DOI: <https://doi.org/10.12720/jomb.1.1.26-29>.
22. Alizadehsani R., Habibi J., Hosseini M. J., Boghrati R., Ghandeharioun A., Bahadorian B., Sani Z. A. Diagnosis of coronary artery disease using data mining techniques based on symptoms and ECG features. *European Journal of Scientific Research*. Vol. 82, No. 4. P. 542–553. 2012. URL: <http://www.europeanjournalofscientificresearch.com>.
23. Державний водний кадастр: облік поверхневих водних об'єктів [Електронний ресурс]. URL: <http://geoportal.davr.gov.ua:81/#hydropostSidebar>.
24. Хільчевський В.К., Лобода Н.С., Ободовський О.Г., Гребінь В.В., Шакірзанова Ж.Р., Ющенко Ю.С., Шерстюк Н.П., Овчарук В.А. Університетська гідрологічна наука в Україні та перспективи подальшого її розвитку. *Український гідрометеорологічний журнал*. Т.19. – С.90 – 105. 2017. URL: <https://uhmj.org.ua/index.php/journal/article/download/74/73>.
25. Шерстюк Н.П., Безуглий В.В. 25 років з часу відновлення геолого-географічного факультету ДНУ: історія становлення та сьогодення. *Вісник Дніпропетровського університету. Серія: геологія, географія*. Т. 24. № 1. С.144–150. 2016. DOI: <https://doi.org/10.15421/111622>.

26. Батурінець А.Г., Антоненко С.В. Огляд програмних засобів для аналізу та візуалізації гідрологічних даних. *Актуальні проблеми автоматизації та інформаційних технологій*. Т. 23. С. 3–14. 2019. DOI: <http://dx.doi.org/10.15421/431901>.
27. Батурінець А.Г., Антоненко С.В. Подовження рядів даних за значеннями показників схожих рядів. *Вісник Черкаського державного технологічного університету*. №3. С.78–86. 2021. ISSN 2708-6070 (Online). DOI: <https://doi.org/10.24025/2306-4412.3.2021.244266>.
28. Baturinets A. Recovery missing values and lengthening the data series. *Austrian Journal of Technical and Natural Sciences*. No. 9–10. P. 3–7. 2021. DOI: <https://doi.org/10.29013/AJT-21-9.10-3-7>.
29. Батурінець А.Г., Антоненко С.В. Визначення схожих гідрологічних рядів даних з використанням коефіцієнтів кореляції. *Системні технології*. Т. 5, № 136. С.98–109. 2021. DOI: <https://doi.org/10.34185/1562-9945-5-136-2021-10>.
30. А. Батурінець, С. Антоненко. Longest common subsequence in the problem of determining the similarity of hydrological data series. *Deutsche Internationale Zeitschrift für zeitgenössische Wissenschaft*. No. 18. 2021. P. 62–64.
31. Suthar B., Patel H., Goswami A. A survey: classification of imputation methods in data mining. *International Journal of Emerging Technology and Advanced Engineering*. Vol. 2, No. 1. P.309-312. 2012.
32. Houari R., Bounceur A., Tari A. K., Kecha M. T. Handling missing data problems with sampling methods. *Proceedings of the 2014 International Conference on Advanced Networking, Distributed Systems and Applications*. IEEE. P. 99–104. 2014. DOI: <https://doi.org/10.1109/INDS.2014.25>.
33. Ayilara O. F., Zhang L., Sajobi T. T., Sawatzky R., Bohm E., Lix L. M. Impact of missing data on bias and precision when estimating change in patient-reported outcomes from a clinical registry. *Health and Quality of Life Outcomes*. Vol. 17, No. 1. P. 106. 2019. DOI: <https://doi.org/10.1186/s12955-019-1181-2>.

34. Kang H. The prevention and handling of the missing data. *Korean Journal of Anesthesiology*. Vol. 64, No. 5. P. 402–406. 2013. DOI: <https://doi.org/10.4097/kjae.2013.64.5.402>.
35. Ludbrook J. Outlying observations and missing values: how should they be handled? *Clinical and Experimental Pharmacology and Physiology*. Vol. 35, No. 5–6. P. 670–678. 2008. DOI: <https://doi.org/10.1111/j.1440-1681.2007.04860.x>.
36. Zhang Z. Missing values in big data research: some basic skills. *Annals of Translational Medicine*. Vol. 3, No. 21. P. 323. 2015. DOI: <https://doi.org/10.3978/j.issn.2305-5839.2015.12.11>.
37. Langkamp D. L., Lehman A., Lemeshow S. Techniques for handling missing data in secondary analyses of large surveys. *Academic Pediatrics*. Vol. 10, No. 3. P. 205–210. 2010. DOI: <https://doi.org/10.1016/j.acap.2010.01.005>.
38. Donders A. R. T., Van Der Heijden G. J., Stijnen T., Moons K. G. Review: a gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*. Vol. 59, No. 10. P. 1087–1091. 2006. DOI: <https://doi.org/10.1016/j.jclinepi.2006.01.014>.
39. Graham J. W. Missing data analysis: making it work in the real world. *Annual Review of Psychology*. Vol. 60. P. 549–576. 2009. DOI: <https://doi.org/10.1146/annurev.psych.58.110405.085530>.
40. Baraldi A. N., Enders C. K. An introduction to modern missing data analyses. *Journal of School Psychology*. Vol. 48, No. 1. P. 5–37, 2010. DOI: <https://doi.org/10.1016/j.jsp.2009.10.001>.
41. Aydilek I. B., Arslan A. A hybrid method for imputation of missing values using optimized fuzzy c-means with support vector regression and a genetic algorithm. *Information Sciences*. Vol. 233, No. 1. P. 25–35. 2013. DOI: <https://doi.org/10.1016/j.ins.2013.01.021>.
42. Lin J., Li N., Alam M. A., Ma Y. Data-driven missing data imputation in cluster monitoring system based on deep neural network. *Applied Intelligence*. Vol. 50, No. 3. P. 860–877. 2020. DOI: <https://doi.org/10.1007/s10489-019-01560-y>.

43. Choudhury A., Kosorok M. R. Missing data imputation for classification problems. *arXiv preprint arXiv:2002.10709*. 2020. DOI: <https://doi.org/10.48550/arXiv.2002.10709>.
44. Al-helali B., Chen Q., Xue B., Zhang M. A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data. *Soft Computing*. Vol. 25, No. 1–2. P. 1–20. 2021. DOI: <https://doi.org/10.1007/s00500-021-05590-y>.
45. Peng D., Zou M., Liu C., Lu J. RESI: A Region-Splitting Imputation method for different types of missing data. *Expert Systems with Applications*. Vol. 168. P. 114425. 2021. DOI: <https://doi.org/10.1016/j.eswa.2020.114425>.
46. Fan J., Han F., Liu H. Challenges of Big Data analysis. *National Science Review*. Vol. 1, No. 2. P. 293–314. 2014. DOI: <https://doi.org/10.1093/nsr/nwt032>.
47. Qiu J., Wu Q., Ding G. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*. No. 67, 2016. DOI: <https://doi.org/10.1186/s13634-016-0355-x>.
48. Rubin D. B. Inference and Missing Data. *Biometrika*. Vol. 63, No. 3. P. 581–592. 1976. DOI: <https://doi.org/10.2307/2335739>.
49. Little R. J. A., Rubin D. B. *Statistical Analysis with Missing Data*. 3rd Edition. Wiley, 2019. 464 p.
50. Graham J. W. *Missing data: Analysis and design*. New York. Springer, 2012. DOI: <https://doi.org/10.1007/978-1-4614-4018-5>.
51. Lin W. C., Tsai C. F. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*. Vol. 53. P. 1487–1509, 2020. DOI: <https://doi.org/10.1007/s10462-019-09709-4>.
52. Buuren S., Groothuis-Oudshoorn C. MICE: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*. Vol. 45, No. 3. 2011. DOI: <https://doi.org/10.18637/jss.v045.i03>.
53. Kamakura W. A., Wedel M. Factor Analysis and Missing Data. *Journal of Marketing Research*. Vol. 37, No. 4. P. 490–498. 2000. DOI: <https://doi.org/10.1509/jmkr.37.4.490.18795>.

54. Shao J., Meng W., Sun G. Evaluation of missing value imputation methods for wireless soil datasets. *Personal and Ubiquitous Computing*. Vol. 21. P. 113–123. 2017. DOI: <https://doi.org/10.1007/s00779-016-0978-9>.
55. Silva-Ramirez E. L., Pino-Mejias R., Lopez-Coello M. Single imputation with multilayer perceptron and multiple imputation combining multilayer perceptron and k-nearest neighbours for monotone patterns. *Applied Soft Computing*. Vol. 29. P. 65–74. 2015. DOI: <https://doi.org/10.1016/j.asoc.2014.09.052>.
56. Tian J., Yu B., Yu D. et al. Missing data analyses: a hybrid multiple imputation algorithm using Gray System Theory and entropy based on clustering. *Applied Intelligence*. Vol. 40. P. 376–388. 2014. DOI: <https://doi.org/10.1007/s10489-013-0469-x>.
57. Pati S. K., Das A. K. Missing value estimation for microarray data through cluster analysis. *Knowledge and Information Systems*. Vol. 52, No. 3. P. 709–750. 2017. DOI: <https://doi.org/10.1007/s10115-017-1025-5>.
58. Twala B. An empirical comparison of techniques for handling incomplete data using decision trees. *Applied Artificial Intelligence*. Vol. 23, No. 5. P. 373–405. 2009. DOI: <https://doi.org/10.1080/08839510902872223>.
59. Xia J., Zhang S., Cai G., Li L., Pan Q., Yan J., Ning G. Adjusted weight voting algorithm for random forests in handling missing values. *Pattern Recognition*. Vol. 69. P. 52–60. 2017. DOI: <https://doi.org/10.1016/j.patcog.2017.04.005>.
60. Zhang Y., Liu Y. Data imputation using least squares support vector machines in urban arterial streets. *IEEE Signal Processing Letters*. Vol. 16, No. 5. P. 414–417. 2009. DOI: <https://doi.org/10.1109/LSP.2009.2016451>.
61. Sefidian A. M., Daneshpour N. Estimating missing data using novel correlation maximization based methods. *Applied Soft Computing*. Vol. 91. No. 106249. 2020. DOI: <https://doi.org/10.1016/j.asoc.2020.106249>.
62. A Barrios A., Trincado G., Garreaud R. Alternative approaches for estimating missing climate data: application to monthly precipitation records in South-Central Chile. *Forest Ecosystems*. Vol. 5, No. 1. P. 1–10. 2018. DOI: <https://doi.org/10.1186/s40663-018-0147-x>.

63. McKnight P. E., McKnight K. M., Sidani S., Figueredo A. J. Missing Data: A Gentle Introduction. *New York: Guilford Press*, 2007. URL: <https://www.guilford.com/excerpts/mcknight.pdf?t=1>.
64. Soley-Bori M. Dealing with missing data: key assumptions and methods for applied analysis. *Boston: Boston University*, 2013. 19 p. URL: https://www.researchgate.net/publication/323266125_Dealing_with_missing_data_key_assumptions_and_methods_for_applied_analysis.
65. Williams R. Missing data Part 1: overview, traditional methods. *Notre Dame: University of Notre Dame*. 2015. 11 p. URL: <https://academicweb.nd.edu/~rwilliam/stats3/MD01.pdf>.
66. Allison P. D. Missing Data. Vol. 136. *New York: Sage Publications*, 2001. DOI: <https://doi.org/10.4135/9781412985079>.
67. Kim J.-O., Curry J. The Treatment of Missing Data in Multivariate Analysis. *Sociological Methods & Research*. Vol. 6, No. 2. P. 215–240. 1977. DOI: <https://doi.org/10.1177/004912417700600206>.
68. Pairwise vs. Listwise deletion: What are they and when should I use them? [Электронный ресурс]. URL: <https://www.ibm.com/support/pages/pairwise-vs-listwise-deletion-what-are-they-and-when-should-i-use-them>.
69. García-Laencina P. J., Sancho-Gómez J., Verleysen M. K nearest neighbours with mutual information for simultaneous classification and missing data imputation. *Neurocomputing*. Vol. 72, No. 7–9. P. 1483–1493. 2009. DOI: <https://doi.org/10.1016/j.neucom.2008.11.026>.
70. Jerez J. M., Molina I., García-Laencina P. J., Alba E., Ribelles N., Martín M., Franco L. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial Intelligence in Medicine*. Vol. 50, No. 2. P. 105–115, 2010. DOI: <https://doi.org/10.1016/j.artmed.2010.05.002>.
71. Khan S. I., Hoque A. S. M. L. SICE: an improved missing data imputation technique. *Journal of Big Data*. Vol. 7, No. 1. P. 1–21. 2020. DOI: <https://doi.org/10.1186/s40537-020-00313-w>.

72. Andridge R. R., Little R. J. A Review of Hot Deck Imputation for Survey Non-Response. *International Statistical Review*. Vol. 78, No. 1. P. 40–64. 2010. DOI: <https://doi.org/10.1111/j.1751-5823.2010.00103.x>.
73. Cheema J. R. A review of missing data handling methods in education research. *Review of Educational Research*. Vol. 84, No. 4. P. 487–508. 2014. DOI: <https://doi.org/10.3102/0034654314532697>.
74. Sullivan D., Andridge R. A hot deck imputation procedure for multiply imputing nonignorable missing data: the proxy pattern-mixture hot deck. *Computational Statistics & Data Analysis*. Vol. 82, No. 1. P. 173–185. 2015. DOI: <https://doi.org/10.1016/j.csda.2014.09.008>.
75. Christopher S. Z., Siswantining T., Sarwinda D., Bustaman A. Missing value analysis of numerical data using fractional hot deck imputation. *3rd International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE, 2019. P. 1–6. DOI: <https://doi.org/10.1109/ICICoS48119.2019.8982412>.
76. Lin W.-C., Tsai C.-F. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*. Vol. 53, No. 2. P. 1487–1509. 2020. DOI: <https://doi.org/10.1007/s10462-019-09709-4>.
77. Rubin L. H., Witkiewitz K., Andre J. S., Reilly S. Methods for handling missing data in the behavioral neurosciences: don't throw the baby rat out with the bath water. *Journal of Undergraduate Neuroscience Education*. Vol. 5, No. 2. P. A71–A77, 2007.
78. Delalleau O., Courville A., Bengio Y. Efficient EM training of gaussian mixtures with missing data. *arXiv preprint arXiv:1209.0521*. 2012. DOI: <https://doi.org/10.48550/arXiv.1209.0521>.
79. Uusitalo L., Lehtikoinen A., Helle I., Myrberg K. An overview of methods to evaluate uncertainty of deterministic models in decision support. *Environmental Modelling & Software*. Vol. 63. P. 24–31. 2015. DOI: <https://doi.org/10.1016/j.envsoft.2014.09.017>.
80. Nguyen C. D., Carlin J. B., Lee K. J. Model checking in multiple imputation: an overview and case study. *Emerging Themes in Epidemiology*. Vol. 14, No. 1. P. 8. 2017. DOI: <https://doi.org/10.1186/s12982-017-0062-6>.

81. Zhao Y., Long Q. Multiple imputation in the presence of high-dimensional data. *Statistical Methods in Medical Research*. Vol. 25, No. 5. P. 2021–2035. 2016. DOI: <https://doi.org/10.1177/0962280213511027>.
82. Huque M. H., Carlin J. B., Simpson J. A., et al. A comparison of multiple imputation methods for missing data in longitudinal studies. *BMC Medical Research Methodology*. Vol. 18. P. 168, 2018. DOI: <https://doi.org/10.1186/s12874-018-0615-6>.
83. Horton N. J., Lipsitz S. R., Parzen M. A potential for bias when rounding in multiple imputation. *The American Statistician*. Vol. 57, No. 4. P. 229–232. 2003. DOI: <https://doi.org/10.1198/0003130032314>.
84. de Goeij M. C., van Diepen M., Jager K. J., Tripepi G., Zoccali C., Dekker F. W. Multiple imputation: dealing with missing data. *Nephrology Dialysis Transplantation*. Vol. 28, No. 10. P. 2415–2420. 2013. DOI: <https://doi.org/10.1093/ndt/gft221>.
85. Khan S. I., Hoque A. S. M. L. SICE: an improved missing data imputation technique. *Journal of Big Data*. Vol. 7, No. 1. P. 1–21. 2020. DOI: <https://doi.org/10.1186/s40537-020-00313-w>.
86. Feature Encoding. *Medium*, 2023. [Электронный ресурс]. URL: <https://medium.com/@denizgunay/feature-encoding-f099a6c1abe8>.
87. LabelEncoder з scikit-learn [Электронный ресурс]. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>.
88. A complete guide on how to handle missing data with IterativeImputer in Python. *Learning AI*, 2023. URL: <https://justlearnai.com/a-complete-guide-on-how-to-handle-missing-data-with-iterativeimputer-in-python-6b224cf0896c>.
89. Imputation of missing values in scikit-learn 1.4.1 [Электронный ресурс]. URL: <https://scikit-learn.org/stable/modules/impute.html>.
90. NoNa: Missing Data Imputation Algorithm. *Medium*, 2023 [Электронный ресурс]. URL: <https://medium.com/@abdualimov/nona-missing-data-imputation-algorithm-d6ff92f70ab8>.

91. Turner-Trauring I. How vectorization speeds up your Python code. *Hyphenated Enterprises LLC*, January 2023. [Електронний ресурс]. URL: <https://pythonspeed.com/articles/vectorization-python/>.
92. Initialization, Finalization, and Threads – Python 2.7.18 documentation. *3.12.3 Documentation* [Електронний ресурс]. URL: <https://docs.python.org/2/c-api/init.html#threads>.
93. concurrent.futures – Launching parallel tasks. *Python documentation*. [Електронний ресурс]. URL: <https://docs.python.org/3/library/concurrent.futures.html>.
94. Chawla N. V., Bowyer K. W., Hall L. O., Kegelmeyer W. P. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*. Vol. 16. P. 321–357. 2002. DOI: <https://doi.org/10.1613/jair.953>.
95. Приставка П. О., Мацуга О. М. Аналіз даних. Навч. посіб. Д.: РВВ ДНУ, 2008. 92 с.
96. IterativeImputer. *Documentation* [Електронний ресурс]. URL: <https://scikit-learn.org/1.5/modules/generated/sklearn.impute.IterativeImputer.html>.
97. XGBoost. *Documentation* [Електронний ресурс]. URL: <https://xgboost.readthedocs.io/en/stable/>.
98. Creates time series features from datetime index. *Kaggle* [Електронний ресурс]. URL: <https://www.kaggle.com/code/robikscube/tutorial-time-series-forecasting-with-xgboost>.
99. Taylor S. J., Letham B. Forecasting at scale. *The American Statistician*. Vol. 72, No. 1. P. 37–45. 2018. URL: <https://peerj.com/preprints/3190.pdf>.
100. Prophet. Python API. *Documentation* [Електронний ресурс]. URL: https://facebook.github.io/prophet/docs/quick_start.html.
101. Forecasting at scale. *Github* [Електронний ресурс]. URL: <https://facebook.github.io/prophet/>.
102. ARIMA model. *Documentation* [Електронний ресурс]. URL: <https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html>.
103. Portilla J. M. Using Python and Auto ARIMA to Forecast Seasonal Time Series [Електронний ресурс]. URL: <https://medium.com/@josemarcialportilla/using-python-and-auto-arima-to-forecast-seasonal-time-series-90877adff03c>.

104. Brownlee J. How to Create an ARIMA Model for Time Series Forecasting in Python [Електронний ресурс]. URL: <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>.

105. Земляний О.Д., Антоненко С.В., Ізмайлова М.К. Методи поповнення пропусків даних гідрологічного моніторингу. *Актуальні проблеми автоматизації та інформаційних технологій*. Т. 24. С. 3–15. 2020. DOI: <http://dx.doi.org/10.15421/432001>.

106. Земляний О.Д., Байбуз О.Г. Огляд методів інтелектуального аналізу даних та методів машинного навчання при прогнозуванні ішемічної хвороби серця. *Актуальні проблеми автоматизації та інформаційних технологій*. Т.27. С. 109–129. 2023. DOI: <http://dx.doi.org/10.15421/432311>.

107. Земляний О.Д., Байбуз О.Г. Методи імпутування пропусків у даних про ішемічну хворобу серця. *Системні технології*. Т. 2, № 151. С.33–49. 2024. DOI: <https://doi.org/10.34185/1562-9945-2-151-2024-04>.

108. Земляний О.Д., Байбуз О.Г. Алгоритми імпутування пропусків у даних на основі ентропії. *Системні технології*. Т. 6, № 155. С. 133–149. 2024. DOI: <https://doi.org/10.34185/1562-9945-6-155-2024-12>.

109. Земляний О.Д., Байбуз О.Г. Імпутування пропусків у даних гідрологічного моніторингу. *Актуальні проблеми автоматизації та інформаційних технологій*. Т. 28. С. 147–160. 2024. DOI: <http://dx.doi.org/10.15421/432413>

110. Земляний О. Д. Розподіли імовірностей випадкових величин у різних гідрологічних задачах. *Тези доповідей XVII Міжнародна науково-практична конференція «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2019)»*, Дніпро, 2019, С. 107. URL: http://mpzis.dnu.dp.ua/wp-content/uploads/2019/12/MPZIS_2019.pdf.

111. Земляний О.Д., Антоненко С.В., Ізмайлова М.К. Пошук підходів до заповнення пропусків даних гідрологічного моніторингу. *Матеріали Міжнародної науково-практичної інтернет-конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*, Вип. 64, Переяслав, 2020, С. 311–312. URL: <https://drive.google.com/file/d/1X3PNALrHe7K6c31HS7yTLtk5kfrNH8KX/>

112. Земляний О.Д., Антоненко С.В., Ізмайлова М.К. Аналіз даних гідрологічного моніторингу водних ресурсів України. *Проблеми прикладної математики та комп'ютерних наук: Тези доповідей тематичної наукової конференції за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара за 2019 рік*, Дніпро, 2020, С. 32–33. URL: <https://repository.dnu.dp.ua/document-details/61992>, <http://repository.dnu.dp.ua:1100/upload/aa8f4fa0e58bf884fe97912d13485911Tezi-2020.pdf>

113. Земляний О.Д., Антоненко С.В., Ізмайлова М.К. Інформаційна технологія в задачах гідрологічного моніторингу. *Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021»*. Хмельницький, 2021. С.97–100. URL: <https://kn.khmn.edu.ua/wp-content/uploads/sites/18/2022/05/3-6-35.pdf>.

114. Oleksii Zemlianyi, Oleh Baibuz. Development of an intelligent monitoring system for making decisions about the state of the cardiovascular system. *Матеріали Міжнародної науково-практичної інтернет-конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*, Вип. 95, Переяслав, 2023. С. 62–63. URL: <https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1>.

115. Земляний О.Д., Байбуз О.Г. Аналіз існуючих методів інтелектуального аналізу даних при прогнозуванні ішемічної хвороби серця. *Тези доповідей XXI Міжнародна науково-практична конференція «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2023)»*. Дніпро, 2023. С.133–134. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf>.

116. Земляний О.Д., Байбуз О.Г. Порівняння багатопроцесорної та багатопоточної реалізацій ентропійного підходу для імпутовання пропусків у даних на мові програмування Python. *Виклики та проблеми сучасної науки*. Т. 2. С. 300–304. 2024. DOI: <https://doi.org/10.15421/cims.2>. URL: <https://cims.fti.dp.ua/j/article/view/131/159>.

117. Земляний О.Д., Байбуз О.Г. Векторизація обчислень для оптимізації коду на мові програмування Python. *Виклики та проблеми сучасної науки*. Т. 3. С. 144–149. 2024. DOI: <https://doi.org/10.15421/cims.3>. URL: <https://cims.fti.dp.ua/j/article/view/215/208>. PURL: <https://purl.org/cims/2403.017>.

118. Земляний О.Д., Байбуз О.Г. Розроблення методів для імпутування пропусків у даних в архітектурі Scikit-learn Python. *Тези доповідей XXII Міжнародна науково-практична конференція «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2024)*. Дніпро, 2024. С.141–143. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf>

119. Земляний О.Д., Байбуз О.Г. Розроблення гібридного методу для імпутування пропусків у даних на основі регресії та ентропійного підходів. *Тези доповідей XXVII Міжнародної конференції «Автоматика 2024»*. Дніпро, 2024. С. 114. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf>

120. Земляний О.Д., Байбуз О.Г. Підвищення якості моделей класифікації та прогнозування шляхом імпутації пропусків. *Тези доповідей XXIII Міжнародна науково-практична конференція «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2025)*. Дніпро, 2025. С. 152–154. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf>

121. The project "Strengthening the standards of teaching, research and international cooperation at Oles Honchar Dnipro National University (DNU)". [Електронний ресурс]. URL: <https://cuni.cz/UK-2184.html>.

ДОДАТКИ

Додаток А

Список праць здобувача за темою дисертації

Статті у наукових фахових виданнях України:

1. Земляний О.Д., Байбуз О.Г. Огляд методів інтелектуального аналізу даних та методів машинного навчання при прогнозуванні ішемічної хвороби серця. *Актуальні проблеми автоматизації та інформаційних технологій*, т. 27, Дніпро, 2023, с. 109–129. DOI: <http://dx.doi.org/10.15421/432311> (особистий внесок Земляного О.Д.: проведення аналізу літературних джерел щодо застосування методів машинного навчання для діагностики ішемічної хвороби серця, систематизація існуючих підходів та алгоритмів, а також підготовка порівняльної характеристики наборів даних, аналіз результатів; Байбуза О.Г.: постановка завдання дослідження, узагальнення отриманих результатів).

2. Земляний О.Д., Байбуз О.Г. Методи імпутування пропусків у даних про ішемічну хворобу серця. *Системні технології. Регіональний міжвузівський збірник наукових праць*, вип. 2(151), Дніпро, 2024, с. 33–49. DOI: <https://doi.org/10.34185/1562-9945-2-151-2024-04> (особистий внесок Земляного О.Д.: розроблення та реалізація алгоритмів імпутування пропусків у медичних даних на основі класифікації та регресії для роботи з категоріальними та кількісними ознаками, проведення експериментального тестування запропонованих методів на двох популярних датасетах, оцінка їхньої точності та швидкодії, аналіз впливу імпутування на якість моделей класифікації, програмування алгоритмів, обробка даних, візуалізація та аналіз результатів; Байбуза О.Г.: постановка мети дослідження, контроль та узагальнення отриманих результатів).

3. Земляний О.Д., Байбуз О.Г. Алгоритми імпутування пропусків у даних на основі ентропії. *Системні технології. Регіональний міжвузівський збірник наукових праць*, вип. 6(155), Дніпро, 2024, с. 133–149. DOI:

<https://doi.org/10.34185/1562-9945-6-155-2024-12> (особистий внесок Земляного О.Д.: розроблення алгоритму *EntropyImputer* на основі ентропійного підходу до імпутування пропусків, реалізація методу мовою *Python*, оптимізація коду для підвищення продуктивності, проведення аналізу ефективності запропонованого методу; Байбуза О.Г.: постановка мети та завдання дослідження, контроль та узагальнення отриманих результатів).

4. Земляний О.Д., Байбуз О.Г. Імпутування пропусків у даних гідрологічного моніторингу. *Актуальні проблеми автоматизації та інформаційних технологій*, т. 28, Дніпро, 2024, с. 147–160. DOI: <http://dx.doi.org/10.15421/432413> (особистий внесок Земляного О.Д.: проведення статистичного аналізу та дослідження особливостей даних гідрологічного моніторингу басейну ріки Дніпро, розроблення методів імпутування, адаптованих для часових рядів, реалізація алгоритмів мовою *Python*, аналіз можливостей щодо оптимізації обчислень, проведення порівняльного аналізу ефективності розроблених методів порівняно з традиційними методами, аналіз результатів; Байбуза О.Г.: постановка задачі, аналіз результатів).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Oleksii Zemlianyi, Oleh Baibuz. Development of an intelligent monitoring system for making decisions about the state of the cardiovascular system. *Матеріали Міжнародної науково-практичної інтернет-конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*, вип. 95, Переяслав, 2023, с. 62–63. URL: <https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1> (особистий внесок Земляного О.Д.: розроблення алгоритмів послідовного аналізу для прийняття рішень щодо категорії захворювання серцево-судинної системи на основі методів перевірки гіпотез про параметри нормального розподілу, реалізація програмного забезпечення

для аналізу даних добового моніторингу артеріального тиску, проведення порівняльного аналізу класичних та послідовних методів; Байбуза О.Г.: постановка завдання, узагальнення отриманих результатів).

6. Земляний О.Д., Байбуз О.Г. Аналіз існуючих методів інтелектуального аналізу даних при прогнозуванні ішемічної хвороби серця. *Тези доповідей XXI Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2023)»*, Дніпро, 22-24 листопада 2023 р., Дніпро, 2023, с. 133–134. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf> (особистий внесок Земляного О.Д.: проведення огляду літератури щодо застосування методів машинного навчання для діагностики ішемічної хвороби серця, систематизація та порівняльний аналіз алгоритмів, аналіз програмних засобів для обробки даних, підготовка порівняльної характеристики наборів даних, аналіз результатів; Байбуза О.Г.: постановка мети і узагальнення отриманих результатів).

7. Земляний О.Д., Байбуз О.Г. Розроблення методів для імпутування пропусків у даних в архітектурі Scikit-learn Python. *Тези доповідей XXII Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2024)»*, Дніпро, 20-22 листопада 2024 р, Дніпро, 2024, с. 141–143. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf> (особистий внесок Земляного О.Д.: розроблення та реалізація класів для імпутування пропусків згідно архітектурних принципів бібліотеки scikit-learn, забезпечення їхньої сумісності з pipeline, тестування на реальних наборах даних, оцінка точності та швидкодії методів, програмування алгоритмів та обробка даних; Байбуза О.Г.: постановка мети і завдання дослідження, контроль результатів).

8. Земляний О.Д., Байбуз О.Г. Розроблення гібридного методу для імпутування пропусків у даних на основі регресії та ентропійного підходів. *Тези доповідей XXVII Міжнародної конференції «Автоматика 2024»*, Дніпро, 20-22 листопада 2024 р., Дніпро, 2024, с.114. URL: <http://mpzis.dnu.dp.ua/wp->

content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf (особистий внесок Земляного О.Д.: розроблення гібридного методу *HybridRegrEntropyImputer*, який поєднує регресійний та ентропійний підходи для імпутування кількісних і якісних ознак, реалізація класу мовою Python згідно архітектурних принципів бібліотеки *scikit-learn*, проведення порівняльного аналізу ефективності методу з іншими розробленими імп'ютерами, оцінка якості імпутування, швидкодії та впливу на зменшення умовної ентропії ознак, аналіз результатів; Байбуза О.Г.: постановка задачі дослідження, аналіз і узагальнення результатів).

9. Земляний О.Д., Байбуз О.Г. Підвищення якості моделей класифікації та прогнозування шляхом імпутації пропусків. *Тези доповідей XXIII Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2025)»*, Дніпро, 19-21 листопада 2025 р., Дніпро: ДНУ, 2025, с.152–154. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf> (особистий внесок Земляного О.Д.: розроблення та реалізація методів імпутування пропусків у даних, аналіз ефективності розроблених методів, інтеграція методів в екосистему бібліотеки *scikit-learn*, проведення апробації на реальних наборах даних, аналіз впливу імпутування на якість моделей класифікації та прогнозування, візуалізація та аналіз результатів; Байбуза О.Г.: постановка мети і завдання дослідження, аналіз і узагальнення отриманих результатів).

Наукові праці в інших виданнях, які додатково відображають зміст дисертації:

10. Земляний О.Д., Байбуз О.Г. Порівняння багатопроцесорної та багатопоточної реалізацій ентропійного підходу для імпутування пропусків у даних на мові програмування Python. *Виклики та проблеми сучасної науки*, т. 2, Дніпро, 2024, с. 300–304. DOI: <https://doi.org/10.15421/cims.2> URL: <https://cims.fti.dp.ua/j/article/view/131/159> (особистий внесок Земляного О.Д.: дослідження підходів щодо оптимізації обчислень при реалізації ентропійного підходу для імпутування пропусків у даних на мові Python, проведення програмного експерименту на реальних наборах даних, дослідження впливу GIL на багатопоточну обробку в Python, аналіз результатів; Байбуза О.Г.: постановка задачі, аналіз результатів).

11. Земляний О.Д., Байбуз О.Г. Векторизація обчислень для оптимізації коду на мові програмування Python. *Виклики та проблеми сучасної науки*, т. 3, Дніпро, 2024, с. 144–149. DOI: <https://doi.org/10.15421/cims.3> URL: <https://cims.fti.dp.ua/j/article/view/215/208>. PURL: <https://purl.org/cims/2403.017> (особистий внесок Земляного О.Д.: дослідження застосування векторизації для підвищення продуктивності та читабельності Python-коду, проведення експериментів на реальних наборах даних, аналіз результатів; Байбуза О.Г.: постановка мети дослідження, аналіз результатів).

Коди класів ім'ютерів та кодувальників якісних ознак

```

import pandas as pd
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin, _fit_context
from sklearn.impute._base import _BaseImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import is_scalar_nan
from sklearn.utils._mask import _get_mask
from sklearn.utils.validation import FLOAT_DTYPES
from tqdm import tqdm

class UnifiedClassRegrImputer(_BaseImputer):
    _parameter_constraints: dict = {
        **_BaseImputer._parameter_constraints,
        "ascending": ["boolean"],
        "sort_columns": ["boolean"],
        "two_steps": ["boolean"],
    }

    def __init__(self,
                 *,
                 regressor=None,
                 classifier=None,
                 sort_columns=False,
                 ascending=False,
                 two_steps=False,
                 verbose=0,
                 missing_values=np.nan,
                 add_indicator=False,
                 keep_empty_features=False, ):
        """
        Unified imputer that handles missing values using different
        strategies:
        - Sort columns by the number of missing values.
        - Two-step approach: handle rows with exactly one missing value
        first.
        - Machine learning-based imputation (regression/classification).

        Parameters:
            regressor: Regression algorithm (default: Ridge with scaling)
            classifier: Classification algorithm (default:
RandomForestClassifier)
            sort_columns: Whether to sort columns by the number of missing
values.
            ascending: Sorting order for columns (True = ascending, False =
descending).
            two_steps: Whether to use two-step approach (first rows with one
missing value).
            verbose: Verbosity level (0 = silent, 1 = show progress bars).
        """
        super().__init__(
            missing_values=missing_values,
            add_indicator=add_indicator,
            keep_empty_features=keep_empty_features,
        )

```

```

        if regressor is None:
            regressor = make_pipeline(StandardScaler(with_mean=False),
Ridge(alpha=0.1))
        if classifier is None:
            classifier = RandomForestClassifier(max_depth=2,
random_state=0)

        self.regressor = regressor
        self.classifier = classifier
        self.sort_columns = sort_columns
        self.ascending = ascending
        self.two_steps = two_steps
        self.verbose = verbose

    @_fit_context(prefer_skip_nested_validation=True)
    def fit(self, X, y=None):
        """Fit the imputer on X.
        Parameters
        -----
        X : array-like shape of (n_samples, n_features)
            Input data, where `n_samples` is the number of samples and
            `n_features` is the number of features.
        y : Ignored
            Not used, present here for API consistency by convention.
        Returns
        -----
        self : object
            The fitted `UnifiedClassRegrImputer` class instance.
        """
        # Check data integrity and calling arguments
        if not is_scalar_nan(self.missing_values):
            force_all_finite = True
        else:
            force_all_finite = "allow-nan"

        X = self._validate_data(
            X,
            accept_sparse=False,
            dtype=FLOAT_DTYPES,
            force_all_finite=force_all_finite,
        )
        self._fit_X = X
        self._mask_fit_X = _get_mask(self._fit_X, self.missing_values)
        self._valid_mask = ~np.all(self._mask_fit_X, axis=0)
        super()._fit_indicator(self._mask_fit_X)
        return self

    def transform(self, X):
        """
        Impute missing values in the DataFrame X.

        Parameters:
            X: DataFrame with missing values

        Returns:
            DataFrame with missing values imputed.
        """
        df = X.copy()

        # Step 1: Impute rows with exactly one missing value if `two_steps`
is True
        if self.two_steps:
            df = self._impute_single_na_rows(df)

```

```

# Step 2: Impute the rest
df = self._impute_remaining(df)

return df

def _impute_single_na_rows(self, df):
    """Impute rows that have exactly one missing value."""
    col_with_1_missing_values = samples_with_count_of_na_is_n(df, n=1,
verbose=self.verbose)
    for i in col_with_1_missing_values: # loop - step 1
        # indexes (rows) with 1 gap, and it's in the i-th column
        index_nan = df[(df.isna().sum(axis=1) == 1) &
            ((df.isnull() @ (df.columns + ",")) .str[:-1] ==
i)].index
        # rows without gaps + rows with the only 1 gap
        data = df.loc[~df.isna().any(axis=1) | df.index.isin(index_nan)]
        # create a train sample for training, it includes only columns
without gaps,
        X_train = data.loc[~data.index.isin(index_nan)]
        if X_train.shape[0] == 0:
            continue
        X_train = X_train.drop([i], axis=1)
        y_train = data.loc[~data.index.isin(index_nan), i]
        # create a test sample, leave the columns in which we don't know
the predicted value
        X_test = data.loc[index_nan]
        X_test = X_test.drop([i], axis=1)
        # if the predicted number is an integer and the number of
predicted values is less than 20,
        # we solve the classification
        if is_column_type_categorical(df[i].unique()):
            method = self.classifier
        else:
            method = self.regressor
        method.fit(X_train, y_train)
        # predict the values and insert them into the missing values in
the column
        df.loc[index_nan, i] = method.predict(X_test)
    return df

def _impute_remaining(self, df):
    """Impute the remaining missing values, optionally sorting columns
first."""
    if self.sort_columns:
        nan_count = df.isna().sum()
        columns = nan_count.sort_values(ascending=self.ascending).index
    else:
        columns = df.columns

    columns = tqdm(columns) if self.verbose == 1 else columns

    for i in columns:
        if df[i].isna().sum() != 0:
            index_nan = df[df[i].isna()].index
            data = df.loc[:, ~df.isna().any()]

            if data.shape[1] == 0:
                df[i] = self._fallback_impute(df, i)
                continue

            X_train = data.loc[~data.index.isin(index_nan)]
            y_train = df[i].loc[~data.index.isin(index_nan)]

```

```

X_test = data.loc[index_nan]

if is_column_type_categorical(df[i].unique()):
    method = self.classifier
else:
    method = self.regressor

method.fit(X_train, y_train)
df.loc[index_nan, i] = method.predict(X_test)

return df

def _fallback_impute(self, df, col):
    """Fallback strategy using median/mode for columns with missing
    values."""
    if pd.api.types.is_numeric_dtype(df[col]):
        return df[col].fillna(df[col].median())
    else:
        return df[col].fillna(df[col].mode()[0])

# Helper functions

def samples_with_count_of_na_is_n(data, n=1, verbose=0):
    """Return list of column names where rows have exactly `n` missing
    values."""
    df2 = data.copy()
    df2['count'] = df2.isnull().sum(axis=1)
    df2['nans'] = (df2.isnull() @ (df2.columns + ",")).str[:-1]
    d = df2[df2['count'] == n]['nans'].tolist()
    return np.unique(d)

def is_column_type_categorical(list_of_unique_values,
    unique_int_values_low_bound=20):
    """Determine if a column type is categorical and should be treated as a
    classification problem."""
    list_not_na = list_of_unique_values[~np.isnan(list_of_unique_values)]
    if len(list_not_na) == 0:
        return False
    return len(np.unique(list_not_na)) < unique_int_values_low_bound and
    all(float(x).is_integer() for x in list_not_na)

```

"""

Параметри: Додамо параметри для керування:
 Використанням сортування стовпців.
 Двоетапною стратегією ім'ютації.
 Використанням регресора або класифікатора для кожного стовпця.

Основні функції класу UnifiedClassRergImputer:

Параметри:

sort_columns: Чи потрібно сортувати стовпці за кількістю пропущених значень.
 ascending: Порядок сортування стовпців за кількістю пропусків (за зростанням
 або спаданням).
 two_steps: Якщо цей параметр активовано, спочатку оброблятимуться рядки з
 одним пропущеним значенням.
 regressor та classifier: Регресійні та класифікаційні алгоритми для
 заповнення пропусків.
 verbose: Рівень деталізації виводу, 0 = без логуювання проміжних розрахунків.

Методи:

`transform`: Основна функція для ім'ютації пропусків.
`_impute_single_na_rows`: Обробляє рядки з одним пропуском.
`_impute_remaining`: Обробляє всі інші пропуски.
`_fallback_impute`: Заповнює стовпці, якщо немає повних стовпців, використовуючи середнє (для числових) або моду (для категоріальних).

Приклад використання:

```
import pandas as pd
from zo_imputers import UnifiedClassRegrImputer

# Приклад набору даних
data = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': ['a', 'b', None, 'a']
})

# Ініціалізація ім'ютера
imputer = UnifiedClassRegrImputer(sort_columns=True, ascending=True,
two_steps=True, verbose=1)
or
imputer = UnifiedClassRegrImputer(sort_columns=True, ascending=False,
two_steps=True, verbose=0)
or
imputer = UnifiedClassRegrImputer(sort_columns=False, two_steps=False,
verbose=0)

# Ім'ютація пропусків
data_imputed = imputer.fit_transform(data)
print(data_imputed)
```

Пояснення:

Сортування стовпців: За кількістю пропусків, щоб обробити стовпці в правильному порядку.
Двоетапна ім'ютація: Спочатку обробляються рядки з одним пропуском, а потім – решта.

Алгоритми: Використовуються регресори для кількісних даних і класифікатори – для категоріальних.

"""

```
import pandas as pd
import numpy as np
from sklearn.base import _fit_context
from sklearn.impute._base import _BaseImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.utils import is_scalar_nan
from sklearn.utils._mask import _get_mask
from sklearn.utils._param_validation import StrOptions, Hidden
from sklearn.utils.validation import FLOAT_DTYPES
from tqdm import tqdm

class RegrImputer(_BaseImputer):
    _parameter_constraints: dict = {
        **_BaseImputer._parameter_constraints,
        "measure": [StrOptions({"value", "weight"}), callable, Hidden(None)],
    }

    def __init__(self,
                 *,
                 regressor=None,
```

```

        measure='value',
        verbose=0,
        missing_values=np.nan,
        add_indicator=False,
        keep_empty_features=False,
    ):
        """
        Imputer that handles missing values using a regressor in two steps:
        - Step 1: Fill rows with only one missing value.
        - Step 2: Fill the rest of the missing values.

        Parameters:
            regressor: Regression algorithm (default: Ridge with scaling)
            measure: 'value' or 'weight', controls how to correct predicted
values for integer columns.
            verbose: Verbosity level (0 = silent, 1 = show progress bar).
        """
        super().__init__(
            missing_values=missing_values,
            add_indicator=add_indicator,
            keep_empty_features=keep_empty_features,
        )
        if regressor is None:
            regressor = make_pipeline(StandardScaler(with_mean=False),
Ridge(alpha=0.1))

        self.regressor = regressor
        self.measure = measure
        self.verbose = verbose

    @_fit_context(prefer_skip_nested_validation=True)
    def fit(self, X, y=None):
        """Fit the imputer on X.
        Parameters
        -----
        X : array-like shape of (n_samples, n_features)
            Input data, where `n_samples` is the number of samples and
            `n_features` is the number of features.
        y : Ignored
            Not used, present here for API consistency by convention.
        Returns
        -----
        self : object
            The fitted `RegrImputer` class instance.
        """
        # Check data integrity and calling arguments
        if not is_scalar_nan(self.missing_values):
            force_all_finite = True
        else:
            force_all_finite = "allow-nan"
        X = self._validate_data(
            X,
            accept_sparse=False,
            dtype=FLOAT_DTYPES,
            force_all_finite=force_all_finite,
        )
        self._fit_X = X
        self._mask_fit_X = _get_mask(self._fit_X, self.missing_values)
        self._valid_mask = ~np.all(self._mask_fit_X, axis=0)
        super()._fit_indicator(self._mask_fit_X)
        return self

    def transform(self, X):

```

```

"""
Impute missing values in the DataFrame X.

Parameters:
    X: DataFrame with missing values

Returns:
    DataFrame with missing values imputed.
"""
df = X.copy()

# Step 1: Impute rows with exactly one missing value
df = self._impute_single_na_rows(df)

# Step 2: Impute the remaining missing values
df = self._impute_remaining(df)

return df

def _impute_single_na_rows(self, df):
    """Impute rows that have exactly one missing value."""
    col_with_1_missing_values = samples_with_count_of_na_is_n(df, n=1,
        verbose=self.verbose)
    for i in col_with_1_missing_values: # loop - step 1
        # indexes (rows) with 1 gap, and it's in the i-th column
        index_nan = df[(df.isna().sum(axis=1) == 1) &
            ((df.isnull() @ (df.columns + ",")).str[:-1] ==
i)].index
        # rows without gaps + rows with the only 1 gap
        data = df.loc[~df.isna().any(axis=1) | df.index.isin(index_nan)]
        # create a train sample for training, it includes only columns
without gaps,
        X_train = data.loc[~data.index.isin(index_nan)]
        if X_train.shape[0] == 0:
            continue
        X_train = X_train.drop([i], axis=1)
        y_train = data.loc[~data.index.isin(index_nan), i]
        # create a test sample, leave the columns in which we don't know
the predicted value
        X_test = data.loc[index_nan]
        X_test = X_test.drop([i], axis=1)
        self.regressor.fit(X_train, y_train)
        Y_test = self.regressor.predict(X_test)
        # predict the values and insert them into the missing values in
the column
        if is_column_type_categorical(df[i].unique()):
            # correct answers
            d = df[i].value_counts().to_dict()
            if self.measure == 'value':
                Y_test = np.vectorize(lambda x: correct_value(d,
x))(Y_test)
            else:
                Y_test = np.vectorize(lambda x: correct_value_weight(d,
x))(Y_test)
        # predict the values and insert them into the missing values in
the column
        df.loc[index_nan, i] = Y_test
    return df

def _impute_remaining(self, df):
    """Impute the remaining missing values."""
    nan_count = df.isna().sum()
    df_columns_sorted = nan_count.sort_values(ascending=False).index

```

```

        columns = tqdm(df_columns_sorted) if self.verbose == 1 else
df_columns_sorted

    for i in columns:
        if df[i].isna().sum() != 0:
            index_nan = df[df[i].isna()].index
            data = df.loc[:, ~df.isna().any()]

            if data.shape[1] == 0:
                df[i] = self._fallback_impute(df, i)
                continue

            X_train = data.loc[~data.index.isin(index_nan)]
            y_train = df[i].loc[~data.index.isin(index_nan)]
            X_test = data.loc[index_nan]

            self.regressor.fit(X_train, y_train)
            Y_test = self.regressor.predict(X_test)

            if is_column_type_categorical(df[i].unique()):
                d = df[i].value_counts().to_dict()
                if self.measure == 'value':
                    Y_test = np.vectorize(lambda x: correct_value(d,
x))(Y_test)
                else:
                    Y_test = np.vectorize(lambda x:
correct_value_weight(d, x))(Y_test)

            df.loc[index_nan, i] = Y_test

    return df

def _fallback_impute(self, df, col):
    """Fallback strategy using median/mode for columns with missing
values."""
    if pd.api.types.is_numeric_dtype(df[col]):
        return df[col].fillna(df[col].median())
    else:
        return df[col].fillna(df[col].mode()[0])

# Helper functions

def samples_with_count_of_na_is_n(data, n=1, verbose=0):
    """Return list of column names where rows have exactly `n` missing
values."""
    df2 = data.copy()
    df2['count'] = df2.isnull().sum(axis=1)
    df2['nans'] = (df2.isnull() @ (df2.columns + ",")).str[:-1]
    d = df2[df2['count'] == n]['nans'].tolist()
    return np.unique(d)

def is_column_type_categorical(list_of_unique_values,
unique_int_values_low_bound=20):
    """Determine if a column type is categorical and should be treated as a
classification problem."""
    list_not_na = list_of_unique_values[~np.isnan(list_of_unique_values)]
    if len(list_not_na) == 0:
        return False
    return len(np.unique(list_not_na)) < unique_int_values_low_bound and
all(float(x).is_integer() for x in list_not_na)

```

```

def correct_value(d, value):
    """Find the closest value based on raw value difference."""
    d2 = {k: (value - k) ** 2 for k, v in d.items()}
    return min(d2, key=d2.get)

def correct_value_weight(d, value):
    """Find the closest value based on weighted difference."""
    n = sum(d.values())
    d2 = {k: (1 - v / n) * ((value - k) ** 2) for k, v in d.items()}
    return min(d2, key=d2.get)

```

"""

План:

Клас RegressorImputer: Ми створимо клас для імп'ютації пропущених значень, що використовує тільки регресор для передбачення відсутніх значень.

Два кроки для імп'ютації: Перший крок – заповнення рядків з одним пропуском, другий – решта пропусків.

Корекція значень: Додамо параметр для корекції цілих значень (через correct_value або correct_value_weight).

Пояснення:

Параметри класу ReprImputer:

regressor: Регресійний алгоритм для передбачення (за замовчуванням – Ridge з масштабуванням).

measure: Стратегія корекції цілих значень – може бути або value (для простого наближення до найближчого значення), або weight (з урахуванням частот значень).

verbose: Рівень деталізації виводу, 0 = без логування проміжних розрахунків.

Два кроки для імп'ютації:

Спочатку заповнюються рядки з одним пропуском, а потім – решта рядків.

Для кожного стовпця, якщо це стовпець із цілими значеннями, використовуються функції correct_value або

correct_value_weight для корекції передбачуваних значень.

Метод _fallback_impute: Якщо немає повних стовпців для тренування, імп'ютер використовує медіану (для кількісних даних) або моду (для категоріальних).

Приклад використання:

```

import pandas as pd
from zo_imputers import ReprImputer

# Приклад набору даних
data = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': ['a', 'b', None, 'a']
})

# Ініціалізація імп'ютера
imputer = ReprImputer(measure='value', verbose=1)

# Імп'ютація пропусків
data_imputed = imputer.fit_transform(data)
print(data_imputed)

```

Основні особливості:

Регресор для всіх типів даних: Імп'ютер використовує регресор навіть для стовпців із категоріальними значеннями,

з подальшою корекцією результатів.
Корекція для категоріальних значень: Реалізовано дві стратегії корекції (value та weight).
Двоетапний підхід: Спочатку обробляються рядки з одним пропуском, а потім всі інші.

```
"""
import pandas as pd
import numpy as np
import concurrent.futures as cf
import math
from scipy.stats import norm
from sklearn.base import _fit_context
from sklearn.impute._base import _BaseImputer
from sklearn.utils import is_scalar_nan
from sklearn.utils._mask import _get_mask
from sklearn.utils._param_validation import StrOptions, Hidden
from sklearn.utils.validation import FLOAT_DTYPES

class EntropyImputer(_BaseImputer):
    _parameter_constraints: dict = {
        **_BaseImputer._parameter_constraints,
        "mode": [StrOptions({"single", "iterative", "threads", "processes"})],
    callable, Hidden(None)],
    }

    def __init__(self,
                 *,
                 target,
                 mode='single',
                 iterations=4,
                 max_workers=None,
                 verbose=0,
                 missing_values=np.nan,
                 add_indicator=False,
                 keep_empty_features=False,
                 ):
        """
        Entropy-based imputer that fills missing values using entropy
        calculations.

        Parameters:
            target: Target column for entropy calculation.
            mode: 'single', 'iterative', 'threads', 'processes' - method of
        imputation.
            iterations: Number of iterations for iterative methods.
            max_workers: Maximum number of workers for thread/process
        execution.
            verbose: Verbosity level (0 = silent, 1 = show details, 2 = show
        more details).
        """
        super().__init__(
            missing_values=missing_values,
            add_indicator=add_indicator,
            keep_empty_features=keep_empty_features,
        )
        self.target = target
        self.mode = mode
        self.iterations = iterations
        self.max_workers = max_workers
        self.verbose = verbose
```

```

@_fit_context(prefer_skip_nested_validation=True)
def fit(self, X, y=None):
    """Fit the imputer on X.
    Parameters
    -----
    X : array-like shape of (n_samples, n_features)
        Input data, where `n_samples` is the number of samples and
        `n_features` is the number of features.
    y : Ignored
        Not used, present here for API consistency by convention.
    Returns
    -----
    self : object
        The fitted `EntropyImputer` class instance.
    """
    # Check data integrity and calling arguments
    if not is_scalar_nan(self.missing_values):
        force_all_finite = True
    else:
        force_all_finite = "allow-nan"
    X = self._validate_data(
        X,
        accept_sparse=False,
        dtype=FLOAT_DTYPES,
        force_all_finite=force_all_finite,
    )
    self._fit_X = X
    self._mask_fit_X = _get_mask(self._fit_X, self.missing_values)
    self._valid_mask = ~np.all(self._mask_fit_X, axis=0)
    super()._fit_indicator(self._mask_fit_X)
    return self

def transform(self, X):
    """
    Impute missing values in the DataFrame X.

    Parameters:
        X: DataFrame with missing values

    Returns:
        DataFrame with missing values imputed.
    """
    df = X.copy()

    if self.mode == 'single':
        return self._fill_entropy_based_1step(df)
    elif self.mode == 'iterative':
        return self._fill_entropy_based_iter(df)
    elif self.mode == 'threads':
        return self._fill_entropy_based_iter_threads(df)
    elif self.mode == 'processes':
        return self._fill_entropy_based_iter_processes(df)
    else:
        raise ValueError(f"Unknown mode: {self.mode}")

def _fill_entropy_based_1step(self, data):
    """Single-step entropy-based imputation."""
    df = data.copy()
    info_data = info(df, self.target)
    col_names = df.columns[df.isna().any()].tolist()
    index_names = ['Info_x_start', 'Info_x_finish']
    df_info_x = pd.DataFrame(columns=col_names, index=index_names)

```

```

for i in col_names:
    method = self._get_method(df[i])
    impute_row_index = df[i].isna().idxmax()
    impute_row_target = df.loc[impute_row_index, self.target]
    value_for_min_info, info_x = (
        method(df[[i, self.target]], i, self.target,
impute_row_target, verbose=self.verbose))
    df_info_x.loc['Info_x_start', i] = info_x
    df.loc[impute_row_index, i] = value_for_min_info
    while df[i].isna().sum() > 0:
        impute_row_index = df[i].isna().idxmax()
        impute_row_target = df.loc[impute_row_index, self.target]
        value_for_min_info, info_x = (
            method(df[[i, self.target]], i, self.target,
impute_row_target, verbose=self.verbose))
        df.loc[impute_row_index, i] = value_for_min_info
        value_for_min_info, info_x = (
            method(df[[i, self.target]], i, self.target,
impute_row_target=None, verbose=self.verbose))
        df_info_x.loc['Info_x_finish', i] = info_x
    if self.verbose >= 1:
        print("df_info_x")
        print(df_info_x)
    return df

def _fill_entropy_based_iter(self, data):
    """Iterative entropy-based imputation."""
    df = data.copy()
    col_names = df.columns[df.isna().any()].tolist()
    df_info_x = pd.DataFrame(columns=col_names, index=['Info_x_start'])

    for i in col_names:
        previous_info_x = float('inf')
        method = self._get_method(df[i])
        _, info_x = method(df[[i, self.target]], i, self.target,
impute_row_target=None, verbose=self.verbose)

        df_info_x.loc['Info_x_start', i] = info_x
        step = 1

        while step <= self.iterations:
            previous_column = df[i].copy()
            df = self._impute_column_iter(df, i, data[i].isna(), method)
            _, info_x = method(df[[i, self.target]], i, self.target,
impute_row_target=None, verbose=self.verbose)

            if info_x < previous_info_x:
                previous_info_x = info_x
                df_info_x.loc['step' + str(step), i] = info_x
            else:
                df[i] = previous_column # Revert if entropy increases
                break
            step += 1

    if self.verbose >= 1:
        print(df_info_x)
    return df

def _fill_entropy_based_iter_threads(self, data):
    """Multi-threaded iterative entropy-based imputation."""
    df = data.copy()
    col_names = df.columns[df.isna().any()].tolist()
    df_info_x = pd.DataFrame(columns=col_names)

```

```

        with cf.ThreadPoolExecutor(max_workers=self.max_workers) as executor:
            futures = [executor.submit(self._task_fill_entropy_based_iter,
                                     df[[i, self.target]], i) for i in col_names]

            for f in cf.as_completed(futures):
                result = f.result()
                df[result[0]] = result[1]
                for elem in result[2]:
                    df_info_x.loc["step" + str(result[2].index(elem)), result[0]]
= elem

            if self.verbose >= 1:
                print(df_info_x)
            return df

    def _fill_entropy_based_iter_processes(self, data):
        """Multi-process iterative entropy-based imputation."""
        df = data.copy()
        col_names = df.columns[df.isna().any()].tolist()
        df_info_x = pd.DataFrame(columns=col_names)

        with cf.ProcessPoolExecutor(max_workers=self.max_workers) as
executor:
            futures = [executor.submit(self._task_fill_entropy_based_iter,
                                     df[[i, self.target]], i) for i in col_names]

            for f in cf.as_completed(futures):
                result = f.result()
                df[result[0]] = result[1]
                for elem in result[2]:
                    df_info_x.loc["step" + str(result[2].index(elem)), result[0]]
= elem

            if self.verbose >= 1:
                print(df_info_x)
            return df

    def _task_fill_entropy_based_iter(self, data, x):
        """Task for a single column in thread/process-based methods."""
        df = data.copy()
        previous_info_x = float('inf')
        method = self._get_method(df[x])
        _, info_x = method(df[[x, self.target]], x, self.target,
impute_row_target=None, verbose=self.verbose)
        info_x_steps = [info_x]

        for _ in range(self.iterations):
            previous_column = df[x].copy()
            df = self._impute_column_iter(df, x, data[x].isna(), method)
            _, info_x = method(df[[x, self.target]], x, self.target,
impute_row_target=None, verbose=self.verbose)

            if info_x < previous_info_x:
                previous_info_x = info_x
                info_x_steps.append(info_x)
            else:
                df[x] = previous_column
                break

        return x, df[x], info_x_steps

    def _impute_column_iter(self, df, col, mask_na, method):

```

```

        """Helper function for iterative imputation of a column."""
        # mask_na = df[col].isna()

        while mask_na.sum() > 0:
            impute_row_index = mask_na.idxmax()
            impute_row_target = df.loc[impute_row_index, self.target]
            value_for_min_info, _ = method(df[[col, self.target]], col,
self.target, impute_row_target,
                                                    verbose=self.verbose)
            df.loc[impute_row_index, col] = value_for_min_info
            mask_na[impute_row_index] = False

        return df

    def _get_method(self, series):
        """Determine whether to use categorical or continuous entropy
calculation."""
        if is_column_type_categorical(series.unique()):
            return info_x_categorical
        else:
            return info_x_continues

def info_x_categorical(df, x, target, impute_row_target=None, verbose=0):
    """
    Entropy calculation for categorical variables.
    df: DataFrame
    x: column for calculating entropy
    target: target column
    impute_row_target: target-value in the row for imputing (optional)
    verbose: verbosity level
    """
    type_map = {'f': float, 'i': int} # 'f' для float, 'i' для int

    type_x = type_map.get(df[x].dtype.kind, str) # Повертає float або int,
або str за замовчуванням
    type_target = type_map.get(df[target].dtype.kind, str)

    data = df.dropna().copy()
    data['conc'] = data[x].astype(str) + '-' + data[target].astype(str)

    # Count occurrences
    freq_vj = data[x].value_counts().to_dict()
    freq_conc_vj_ci = data['conc'].value_counts().to_dict()

    # Calculate entropy components
    info_x_vj = {vj: 0 for vj in freq_vj.keys()}
    total_count = sum(freq_vj.values()) + (1 if impute_row_target else 0)

    for key, count in freq_conc_vj_ci.items():
        vj, ci = key.split('-')
        vj = type_x(vj)
        ci = type_target(ci)
        freq = count
        if impute_row_target is not None:
            if ci == impute_row_target: # df_index_target:
                freq += 1
        prob = freq / total_count
        info_x_vj[vj] += -prob * math.log2(prob)

    value_for_min_info = min(info_x_vj, key=info_x_vj.get)
    # info_x = sum((count / total_count) * info for vj, info in
info_x_vj.items() for count in [freq_vj.get(vj, 0)])

```

```

info_x = 0
s = sum(freq_vj.values())
for key, count in freq_vj.items():
    probability = count / s
    info_x += probability * info_x_vj[key]

if verbose == 2:
    print("Categorical Entropy info_x_vj(", x, ")=", info_x_vj)

return value_for_min_info, info_x

def info_x_continues(df, x, target, impute_row_target=None, verbose=0):
    """
    Entropy calculation for continuous variables using discretization.
    df: DataFrame
    x: column for calculating entropy
    target: target column
    impute_row_target: target-value in the row for imputing (optional)
    verbose: verbosity level
    """
    data = df.dropna().copy()

    # Discretize using equal-width intervals
    m = int(1 + 3.32 * math.log10(data.shape[0]))
    min_x = data[x].min()
    max_x = data[x].max()
    h = (max_x - min_x) / m
    #  $x_l = x_{min} + l * h, l = 0..m$ 
    intervals = [min_x + i * h for i in range(0, m)]
    intervals.append(intervals[-1] + h + 0.001 * h)
    data.loc[:, "interval"] = np.searchsorted(intervals, data[x],
side='right')
    if verbose == 2:
        print("column: ", x)
        print("min_x= ", min_x)
        print("max_x= ", max_x)
        print("intervals = ", intervals)
        print(data)

    freq_interval = data["interval"].value_counts().to_dict()
    freq_interval = dict(sorted(freq_interval.items()))

    sum_interval = data.groupby(["interval"])[x].sum().to_dict()

    mean_interval = {}
    for k, v in sum_interval.items():
        mean_interval[k] = sum_interval[k] / freq_interval[k]

    data.loc[:, "sqr(x-x_sr)"] = (data[x] - data["interval"].apply(lambda y:
mean_interval[y])) ** 2
    std_interval = data.groupby(["interval"])["sqr(x-x_sr)"].sum().to_dict()
    for k, v in sum_interval.items():
        std_interval[k] = math.sqrt(std_interval[k] / freq_interval[k])

    # Recalculate using categorical method for intervals
    value_for_min_info_interval, info_x = (
        info_x_categorical(data[["interval", target]],
                           "interval", target, impute_row_target,
verbose=verbose))
    # generate value from N(m, s)
    # which belongs to the (value_for_min_info_interval)-th interval
    if std_interval[value_for_min_info_interval] > 0:

```

```

        value_for_min_info = norm.rvs(size=1,
loc=mean_interval[value_for_min_info_interval],
scale=std_interval[value_for_min_info_interval])
    else:
        value_for_min_info = mean_interval[value_for_min_info_interval]
    return value_for_min_info, info_x

def info(df, target):
    """
    General information entropy for the target column.
    df: DataFrame
    target: target column
    """
    d = df[target].value_counts().to_dict()
    info_dict = {k: -v / sum(d.values()) * math.log2(v / sum(d.values())) for
k, v in d.items()}
    return sum(info_dict.values())

# Helper functions

def is_column_type_categorical(list_of_unique_values,
unique_int_values_low_bound=15):
    """Determine if a column type is categorical and should be treated as a
categorical column."""
    list_not_na = list_of_unique_values[~np.isnan(list_of_unique_values)]
    return len(list_not_na) > 0 and len(np.unique(list_not_na)) <
unique_int_values_low_bound and all(
    float(x).is_integer() for x in list_not_na)

"""
Основні елементи класу EntropyImputer:
Параметри:

target: Рядок з назвою цільового стовпця.
mode: Визначає тип ітерації: single (одноразова імп'ютація), iterative,
threads, або processes.
iterations: Кількість ітерацій для ітераційних методів.
max_workers: Максимальна кількість робітників для багатопотокових або
багатопроцесних варіантів.
verbose: Рівень деталізації виводу, 0 = без логування проміжних розрахунків.
Методи для кожного режиму: Клас надає варіанти для одноразової, ітераційної,
багатопотокової та багатопроцесної
імп'ютації.

Використання:
import pandas as pd
from entropy_imputer import EntropyImputer

# Приклад набору даних
data = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': ['a', 'b', None, 'a'],
    'target': [1, 0, 1, 0]
})

# Ініціалізація імп'ютера з багатопотоковою імп'ютацією

```

```

imputer = EntropyImputer(target='target', mode='threads', verbose=1)

# Імп'ютація пропусків
data_imputed = imputer.fit_transform(data)
print(data_imputed)

"""

import pandas as pd
import numpy as np
from sklearn.base import _fit_context
from sklearn.impute._base import _BaseImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.utils import is_scalar_nan
from sklearn.utils._mask import _get_mask
from sklearn.utils.validation import FLOAT_DTYPES
from tqdm import tqdm

from zo_imputers import EntropyImputer

class HybridRegrEntropyImputer(_BaseImputer):
    _parameter_constraints: dict = {
        **_BaseImputer._parameter_constraints,
    }

    def __init__(self,
                 *,
                 target,
                 regressor=None,
                 verbose=0,
                 missing_values=np.nan,
                 add_indicator=False,
                 keep_empty_features=False,
                 ):
        """
        Imputer that handles missing values using a regressor in two steps:
        - Step 1: Fill rows with only one missing value.
        - Step 2: Fill the rest of the missing values.

        Parameters:
            regressor: Regression algorithm (default: Ridge with scaling)
            verbose: Verbosity level (0 = silent, 1 = show progress bar).
        """
        super().__init__(
            missing_values=missing_values,
            add_indicator=add_indicator,
            keep_empty_features=keep_empty_features,
        )
        if regressor is None:
            regressor = make_pipeline(StandardScaler(with_mean=False),
                                     Ridge(alpha=0.1))

        self.target = target
        self.regressor = regressor
        self.verbose = verbose

    @fit_context(prefer_skip_nested_validation=True)
    def fit(self, X, y=None):
        """Fit the imputer on X.

```

```

Parameters
-----
X : array-like shape of (n_samples, n_features)
    Input data, where `n_samples` is the number of samples and
    `n_features` is the number of features.
y : Ignored
    Not used, present here for API consistency by convention.
Returns
-----
self : object
    The fitted `RegrImputer` class instance.
"""
# Check data integrity and calling arguments
if not is_scalar_nan(self.missing_values):
    force_all_finite = True
else:
    force_all_finite = "allow-nan"
X = self._validate_data(
    X,
    accept_sparse=False,
    dtype=FLOAT_DTYPES,
    force_all_finite=force_all_finite,
)
self._fit_X = X
self._mask_fit_X = _get_mask(self._fit_X, self.missing_values)
self._valid_mask = ~np.all(self._mask_fit_X, axis=0)
super()._fit_indicator(self._mask_fit_X)
return self

def transform(self, X):
    """
    Impute missing values in the DataFrame X.

    Parameters:
        X: DataFrame with missing values

    Returns:
        DataFrame with missing values imputed.
    """
    df = X.copy()

    # impute the missing values in the numerical columns
    # Step 1: Impute rows with exactly one missing value
    df = self._impute_single_na_rows(df)
    # Step 2: Impute the remaining missing values
    df = self._impute_remaining(df)

    # Step 3: Impute the missing values in the categorical columns
    df = EntropyImputer(target=self.target, mode='single',
verbose=self.verbose).transform(df)

    return df

def _impute_single_na_rows(self, df):
    """Impute rows that have exactly one missing value."""
    col_with_1_missing_values = samples_with_count_of_na_is_n(df, n=1)
    for i in col_with_1_missing_values: # loop - step 1
        if is_column_type_categorical(df[i].unique()):
            continue
        # indexes (rows) with 1 gap, and it's in the i-th column
        index_nan = df[(df.isna().sum(axis=1) == 1) &
            ((df.isnull() @ (df.columns + ",")) .str[:-1] ==
i)].index

```

```

        # rows without gaps + rows with the only 1 gap
        data = df.loc[~df.isna().any(axis=1) | df.index.isin(index_nan)]
        # create a train sample for training, it includes only columns
without gaps,
        X_train = data.loc[~data.index.isin(index_nan)]
        if X_train.shape[0] == 0:
            continue
        X_train = X_train.drop([i], axis=1)
        y_train = data.loc[~data.index.isin(index_nan), i]
        # create a test sample, leave the columns in which we don't know
the predicted value
        X_test = data.loc[index_nan]
        X_test = X_test.drop([i], axis=1)
        self.regressor.fit(X_train, y_train)
        Y_test = self.regressor.predict(X_test)
        # predict the values and insert them into the missing values in
the column
        df.loc[index_nan, i] = Y_test
    return df

    def _impute_remaining(self, df):
        """Impute the remaining missing values."""
        nan_count = df.isna().sum()
        df_columns_sorted = nan_count.sort_values(ascending=False).index

        columns = tqdm(df_columns_sorted) if self.verbose == 1 else
df_columns_sorted

        for i in columns:
            if is_column_type_categorical(df[i].unique()):
                continue
            if df[i].isna().sum() != 0:
                index_nan = df[df[i].isna()].index
                data = df.loc[:, ~df.isna().any()]

                if data.shape[1] == 0:
                    df[i] = self._fallback_impute(df, i)
                    continue

                X_train = data.loc[~data.index.isin(index_nan)]
                y_train = df[i].loc[~data.index.isin(index_nan)]
                X_test = data.loc[index_nan]

                self.regressor.fit(X_train, y_train)
                Y_test = self.regressor.predict(X_test)

                df.loc[index_nan, i] = Y_test

        return df

    def _fallback_impute(self, df, col):
        """Fallback strategy using median/mode for columns with missing
values."""
        if pd.api.types.is_numeric_dtype(df[col]):
            return df[col].fillna(df[col].median())
        else:
            return df[col].fillna(df[col].mode()[0])

# Helper functions
def samples_with_count_of_na_is_n(data, n=1):
    """Return list of column names where rows have exactly `n` missing

```

```

values. """
    df2 = data.copy()
    df2['count'] = df2.isnull().sum(axis=1)
    df2['nans'] = (df2.isnull() @ (df2.columns + ",")).str[:-1]
    d = df2[df2['count'] == n]['nans'].tolist()
    return np.unique(d)

def is_column_type_categorical(list_of_unique_values,
unique_int_values_low_bound=20):
    """Determine if a column type is categorical and should be treated as a
classification problem."""
    list_not_na = list_of_unique_values[~np.isnan(list_of_unique_values)]
    if len(list_not_na) == 0:
        return False
    return len(np.unique(list_not_na)) < unique_int_values_low_bound and
all(float(x).is_integer() for x in list_not_na)

"""
План:
Клас RegressorImputer: Ми створимо клас для ім'ютації пропущених значень, що
використовує тільки регресор для
передбачення відсутніх значень.
Два кроки для ім'ютації: Перший крок – заповнення рядків з одним пропуском,
другий – решта пропусків.
Корекція значень: Додамо параметр для корекції цілих значень (через
correct_value або correct_value_weight).

Пояснення:
Параметри класу RegrImputer:

regressor: Регресійний алгоритм для передбачення (за замовчуванням – Ridge з
масштабуванням).
measure: Стратегія корекції цілих значень – може бути або value (для простого
наближення до найближчого значення),
або weight (з урахуванням частот значень).
verbose: Рівень деталізації виводу, 0 = без логування проміжних розрахунків.
Два кроки для ім'ютації:

Спочатку заповнюються рядки з одним пропуском, а потім – решта рядків.
Для кожного стовпця, якщо це стовпець із цілими значеннями, використовуються
функції correct_value або
correct_value_weight для корекції передбачуваних значень.
Метод _fallback_impute: Якщо немає повних стовпців для тренування, ім'ютер
використовує медіану (для кількісних даних)
або моду (для категоріальних).

Приклад використання:
import pandas as pd
from zo_imputers import RegrImputer

# Приклад набору даних
data = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': ['a', 'b', None, 'a']
})

# Ініціалізація ім'ютера
imputer = HybridRegrEntropyImputer(target='target', verbose=1)

# Ім'ютація пропусків

```

```
data_imputed = imputer.fit_transform(data)
print(data_imputed)
```

Основні особливості:

Регресор для всіх типів даних: Імп'ютер використовує регресор навіть для стовпців із кількісними значеннями, з подальшою корекцією результатів.

Корекція для цілих значень: Реалізовано дві стратегії корекції для кількісних значень (value та weight).

Двоетапний підхід: Спочатку обробляються рядки з одним пропуском, а потім всі інші.

```
"""
```

```
import numpy as np
import pandas as pd
import scipy.stats as ss
from matplotlib import pyplot as plt, cm
from sklearn.base import _fit_context
from sklearn.impute._base import _BaseImputer
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.utils import is_scalar_nan
from sklearn.utils._mask import _get_mask
from sklearn.utils.validation import FLOAT_DTYPES
```

```
class CorrelationImputer(_BaseImputer):
```

```
    _parameter_constraints: dict = {
        **_BaseImputer._parameter_constraints,
    }
```

```
    def __init__(self,
                 *,
                 column,
                 target,
                 alpha=0.05,
                 verbose=0,
                 visual_each=0,
                 visual_together=0,
                 missing_values=np.nan,
                 add_indicator=False,
                 keep_empty_features=False,
                 ):
        """
```

```
        Impute missing values in one column using another correlated column
        based on quasi-linear regression.
```

```
        Perform imputation on `target` based on `column` using quasi-linear
        regression models.
```

```
        Parameters:
```

```
        column: Independent variable
```

```
        target: Column to be filled in
```

```
        alpha: Significance level for regression analysis.
```

```
        verbose: Verbosity level (0 = silent, 1 = brief, 2 = detailed).
```

```
        visual_each: If True, plots the regression and confidence
```

```
        intervals for each model.
```

```
        visual_together: If True, plots all regression lines together.
```

```
        """
```

```
        super().__init__(
            missing_values=missing_values,
            add_indicator=add_indicator,
            keep_empty_features=keep_empty_features,
```

```

    )
    self.column = column
    self.target = target
    self.alpha = alpha
    self.verbose = verbose
    self.visual_each = visual_each
    self.visual_together = visual_together

@_fit_context(prefer_skip_nested_validation=True)
def fit(self, X, y=None):
    """Fit the imputer on X.
    Parameters
    -----
    X : array-like shape of (n_samples, n_features)
        Input data, where `n_samples` is the number of samples and
        `n_features` is the number of features.
    y : Ignored
        Not used, present here for API consistency by convention.
    Returns
    -----
    self : object
        The fitted `CorrelationImputer` class instance.
    """
    # Check data integrity and calling arguments
    if not is_scalar_nan(self.missing_values):
        force_all_finite = True
    else:
        force_all_finite = "allow-nan"

    X = self._validate_data(
        X,
        accept_sparse=False,
        dtype=FLOAT_DTYPES,
        force_all_finite=force_all_finite,
    )
    self._fit_X = X
    self._mask_fit_X = _get_mask(self._fit_X, self.missing_values)
    self._valid_mask = ~np.all(self._mask_fit_X, axis=0)
    super()._fit_indicator(self._mask_fit_X)
    return self

def transform(self, X):
    """
    Impute missing values in a DataFrame.

    Parameters:
        X: DataFrame with missing values to be imputed.

    Returns:
        DataFrame with imputed values.
    """
    df = X.copy()
    df = self._transform_1_step(df, self.column, self.target)
    df = self._transform_1_step(df, self.target, self.column)

    return df

def _transform_1_step(self, X, column, target):
    df = X.copy()
    self._num_picture = 1
    # Drop rows where both columns have NaN
    df_clean = df.dropna(subset=[column, target])
    x = df_clean[column]

```

```

y = df_clean[target]
a, b, f = self.get_best_regression(x, y, column, target)
# Impute missing values in column2 based on column1
X[target] = X[target].fillna(f(a, b, X[column]))
return X

def get_best_regression(self, x, y, column, target):
    """Select the best regression model sequentially."""
    col_names = ['a', 'b', 'R2', 'mse', 'sign', 'f']
    index_names = [
        'model 1: y = a + b * x',
        'model 2: y = a + b * ln(x)',
        'model 3: y = exp(a + b * x)',
        'model 4: y = exp(a + b * ln(x))',
        'model 5: y = ln(a + b * x)',
        'model 6: y = a + b / x',
        'model 7: y = a + b * e^x',
        'model 8: y = sqrt(a + b * x)'
    ]
    df_regr_info = pd.DataFrame(columns=col_names, index=index_names)

    models = [
        (regr_a_plus_b_x, x, y, "model 1: y = a + b * x"),
        (regr_a_plus_b_log_x, np.log(x), y, "model 2: y = a + b *
ln(x)"),
        (regr_exp_a_plus_b_x, x, np.log(y), "model 3: y = exp(a + b *
x)"),
        (regr_exp_a_plus_b_log_x, np.log(x), np.log(y), "model 4: y =
exp(a + b * ln(x))"),
        (regr_log_a_plus_b_x, x, np.exp(y), "model 5: y = ln(a + b *
x)"),
        (regr_a_plus_b_div_x, 1 / x, y, "model 6: y = a + b / x"),
        (regr_a_plus_b_exp_x, np.exp(x), y, "model 7: y = a + b * e^x"),
        (regr_sqrt_a_plus_b_x, x, y ** 2, "model 8: y = sqrt(a + b * x)")
    ]

    for f, x_transformed, y_transformed, text in models:
        self._set_regr_info(x_transformed, y_transformed, x, y, f,
df_regr_info, text)

        # Find the best model
        best_model_mse = df_regr_info['mse'].idxmin()
        a_best = df_regr_info.loc[best_model_mse, 'a']
        b_best = df_regr_info.loc[best_model_mse, 'b']
        f_best = df_regr_info.loc[best_model_mse, 'f']

        if self.verbose > 0:
            pd.set_option("display.max_columns", None)
            print(df_regr_info[['a', 'b', 'R2', 'mse']])
            print("best: ", best_model_mse, " ", a_best, " ", b_best, " ",
f_best)

        if self.visual_together == 1:
            plt.figure(figsize=(10, 5))
            color = iter(cm.rainbow(np.linspace(0, 1, df_regr_info.shape[0] +
1)))

            c = next(color)
            plt.plot(x, y, 'o', color=c, label="Correlation field")

            x_s = x.sort_values()
            for idx, row in df_regr_info.iterrows():
                y_new = row['f'](row['a'], row['b'], x_s)
                c = next(color)

```

```

        plt.plot(x_s, y_new, color=c, label=idx)

        plt.xlabel("x - " + column)
        plt.ylabel("y - " + target)
        plt.grid()
        plt.title('Correlation field and linear regressions')
        plt.legend()

    if self.visual_each == 1 or self.visual_together == 1:
        plt.show()

    return a_best, b_best, f_best

def _set_regr_info(self, x, y, x0, y0, f, df_regr_info, text):
    """Helper function to compute regression information."""
    """
    :param x: x after transformation to the linear
    :param y: y after transformation to the linear
    :param x0: base x
    :param y0: base y
    :param column: column name for independent variable
    :param target: column name for dependent variable
    :param f: real linear or quasi-linear function to calculate mse in
the base scale
    :param regr_info: df with regression information
    :return: nothing
    """
    if self.verbose > 0:
        print(text)
    if self.verbose == 2 or self.visual_each == 1:
        self._analysis2D(x, y, text=text)
    R2 = self._R2(x, y)
    n = len(y)
    _, _, sign_R2 = self._linear_sign(R2, n, 2)
    a, b = self._linearModel2D(x, y)
    y1 = f(a, b, x0)
    mse_resid = self._mse(y0, y1, 2)
    if self.verbose == 1:
        print("mse_resid in the base scale = {:.3f}".format(mse_resid))

    if df_regr_info is not None:
        df_regr_info.loc[text, 'a'] = a
        df_regr_info.loc[text, 'b'] = b
        df_regr_info.loc[text, 'mse'] = mse_resid
        df_regr_info.loc[text, 'R2'] = R2
        df_regr_info.loc[text, 'sign'] = sign_R2
        df_regr_info.loc[text, 'f'] = f

def _analysis2D(self, x, y, text=""):
    n = len(y)

    a0, a1 = self._linearModel2D(x, y)
    f = regr_a_plus_b_x(a0, a1, x)

    D_a0, D_a1 = self._DC_2D(x, y, f)
    # residual variance
    mse_resid = self._mse(y, f, 2)

    confident_probability = 1 - self.alpha # =0.95
    # t - Student's distribution
    # t_alfa = ss.t.ppf((1 + 0.95)/2, n-2)
    t_alpha = ss.t.ppf((1 + confident_probability) / 2, n - 2)
    # variance D(f(x))

```

```

D_f = mse_resid / n + D_a1 * ((x - x.mean()) ** 2)
# confidence interval for the regression
f_l = f - t_alpha * np.sqrt(D_f)
f_r = f + t_alpha * np.sqrt(D_f)
# confidence interval on the forecast value
y_l = f - t_alpha * np.sqrt(D_f + mse_resid)
y_r = f + t_alpha * np.sqrt(D_f + mse_resid)

l_a0, r_a0 = self._param_interval(a0, D_a0, n)
l_a1, r_a1 = self._param_interval(a1, D_a1, n)
t_a0, t_alfa_a0, sign_a0 = self._param_sign(a0, D_a0, n)
t_a1, t_alfa_a1, sign_a1 = self._param_sign(a1, D_a1, n)

R2 = self._R2(x, y)
f_stat, f_alpha, sign_R2 = self._linear_sign(R2, n, 2)

residuals = y - f

if self.verbose == 2:
    print('%24s\t%8s\t%16s\t%8s\t%8s\t%8s' % (
        'Estimate', 'SSE', 'Conf.int.', 'Statistics', 'Quantile',
'Sign/Insign'))
    print('%8s\t%8.3f\t%8.3f\t[%6.3f, %6.3f]\t%8.3f\t%8.3f\t%8s' % (
        'a0', a0, D_a0, l_a0, r_a0, t_a0, t_alfa_a0, sign_a0))
    print('%8s\t%8.3f\t%8.3f\t[%6.3f, %6.3f]\t%8.3f\t%8.3f\t%8s' % (
        'a1', a1, D_a1, l_a1, r_a1, t_a1, t_alfa_a1, sign_a1))

    print('%8s\t%8.3f\t%32s\t%8.3f\t%8.3f\t%8s' % ('R2', R2, "",
f_stat, f_alpha, sign_R2))
    print("Determination coefficient R2 = {:.3f}".format(R2))

    print("Residual variance mse_resid = {:.3f}".format(mse_resid))

    stat, p = ss.shapiro(residuals)
    if p > self.alpha:
        print("A normal distribution is confirmed for the residuals")
    else:
        print("A normal distribution isn't confirmed for the
residuals")

if self.visual_each == 1:
    plt.figure(figsize=(20, 4))
    plt.subplot(1, 2, 1)
    plt.hist(residuals)
    plt.title("Residuals histogram")
    plt.subplot(1, 2, 2)
    plt.plot(x, y, 'o', color="g", label="Correlation field")
    plt.plot(x, f, color="r", label="Regression")
    plt.plot(x, f_l, color="c", label="Conf.interval on the
regression")
    plt.plot(x, f_r, color="c")
    plt.plot(x, y_l, color="y", label="Conf.interval on the
forecast")
    plt.plot(x, y_r, color="y")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title('Correlation field and linear regression ' + text)
    plt.legend(loc="best")

def _linearModel2D(self, x, y):
    # univariate linear regression parameters
    S_x = np.std(x)
    S_y = np.std(y)

```

```

a1 = ss.pearsonr(x, y)[0] * S_y / S_x
a0 = y.mean() - a1 * x.mean()
return a0, a1

def _linear_sign(self, R2, n, p=2):
    # Testing the significance of linear regression
    # R2 - determination coefficient
    # N - sample size
    # p - parameter count
    f = R2 * (n - p - 1) / (1 - R2) / p
    confident_probability = 1 - self.alpha # =0.95
    # f - Fisher's distribution
    # f_alfa = ss.f.ppf(1 - 0.05, p, N-p-1)
    f_alpha = ss.f.ppf(confident_probability, p, n - p - 1)
    if f <= f_alpha:
        # insignificant
        result = False
    else:
        # significant
        result = True
    return f, f_alpha, result

def _SSE(self, y, f):
    # sum of squared residuals
    # y - initial array
    # f - array calculated by the regression model
    return sum((y - f) ** 2)

def _mse(self, y, f, s=2):
    # y - initial array
    # f - array calculated by the regression model
    # s = 2 - number of parameters including the free term
    N = len(y)
    return self._SSE(y, f) / (N - s)

def _DC_2D(self, x, y, f):
    # estimation variance
    N = len(y)
    S_x = np.std(x)
    mse_resid = self._mse(y, f, 2)
    multiplier = mse_resid / ((N ** 2) * (S_x ** 2))
    D_a0 = sum(x * x) * multiplier
    D_a1 = N * multiplier
    return D_a0, D_a1

def _R2(self, x, y):
    return (ss.pearsonr(x, y)[0]) ** 2

def _param_interval(self, param, D_param, n):
    # confidence interval for parameter estimation
    # param - parameter estimate
    # D_param - variance of the estimate
    # n - sample length
    alpha = 0.05
    confident_probability = 1 - alpha # =0.95
    # t - Student's distribution
    # t_alfa = ss.t.ppf((1 + 0.95)/2, n-2)
    t_alpha = ss.t.ppf((1 + confident_probability) / 2, n - 2)
    left = param - t_alpha * np.sqrt(D_param)
    right = param + t_alpha * np.sqrt(D_param)
    return left, right

def _param_sign(self, param, D_param, n):

```

```

# Checking the parameter for equality 0 (insignificance)
# param - parameter estimate
# D_param - variance of estimation
# n - sample length
alpha = 0.05
confident_probability = 1 - alpha # =0.95
# t - Student's distribution
# t_alfa = ss.t.ppf((1 + 0.95)/2, n-2)
t_alfa = ss.t.ppf((1 + confident_probability) / 2, n - 2)
t = param / np.sqrt(D_param)
if np.abs(t) <= t_alfa:
    result = "insignificant"
else:
    result = "significant"
return t, t_alfa, result

# Helper quasi-linear functions

def regr_a_plus_b_x(a0, a1, x):
    # 1: linear regression
    return a0 + a1 * x

def regr_a_plus_b_log_x(a, b, x):
    # 2 - linear transform x1 = np.log(x)
    return a + b * np.log(x)

def regr_exp_a_plus_b_x(a, b, x):
    # 3 - linear transform y1 = np.log(y)
    return np.exp(a + b * x)

def regr_exp_a_plus_b_log_x(a, b, x):
    # 4 - linear transform x1 = np.log(x), y1 = np.log(y)
    return np.exp(a + b * np.log(x))

def regr_log_a_plus_b_x(a, b, x):
    # 5 - linear transform y1 = np.exp(y)
    return np.log(a + b * x)

def regr_a_plus_b_div_x(a, b, x):
    # 6 - linear transform x1 = 1 / x)
    return a + b / x

def regr_a_plus_b_exp_x(a, b, x):
    # 7 - linear transform x1 = np.exp(x)
    return a + b * np.exp(x)

def regr_sqrt_a_plus_b_x(a, b, x):
    # 8 - linear transform y1 = y * y
    return np.sqrt(a + b * x)

"""
План:
Клас CorrelationImputer: Ми створимо клас для ім'ютації пропущених значень,
що використовує інформацію про наявність

```

кореляційного зв'язку для передбачення відсутніх значень однієї ознаки за даними з іншої ознаки за знайденою найкращою регресійною залежністю.

Пояснення:

Параметри класу `CorrelationImputer`:

`column`: незалежна ознака
`target`: залежна ознака
`alpha`: критичний рівень значущості для регресійного аналізу, за замовчуванням =0.05
`verbose`: Рівень деталізації виводу, 0 = без логування проміжних розрахунків, 1 = коротко, 2 = детально
`visual_each`: Рівень деталізації для графіків залежностей, 0 = без графіків, 1 = виводяться графіки для кожної залежності окремо
`visual_together`: Рівень деталізації для спільного графіку для всіх розглянутих залежностей, 0 = без графіку

Приклад використання:

```
import pandas as pd
from correlation_imputer import CorrelationImputer

# Приклад набору даних
data = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [2, None, 6, 8, None]
})

# Ініціалізація ім'ютера
imputer = CorrelationImputer(column='A', target='B', verbose=1,
                              visual_each=1)

# Налаштувати ім'ютер на набір даних
imputer.fit(df)

# Імп'ютація стовпця B на основі значень стовпця A
imputed_data = imputer.transform(df)
print(imputed_data)

"""

import pandas as pd
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import LabelEncoder

class IgnoreNaNLabelEncoder:
    def __init__(self):
        self.encoders = {}

    def fit(self, X):
        """
        Fits label encoders for each object/category column in the DataFrame.

        Parameters:
        X (pd.DataFrame): Input DataFrame with categorical columns.

        Returns:
        self: Fitted transformer.
        """
        for col_name in X.select_dtypes(include=['object'],
```

```

'category'])).columns:
    series = X[col_name]
    label_encoder = LabelEncoder()

    # Fit on non-null values only
    label_encoder.fit(series.dropna())

    # Store the label encoder for each column
    self.encoders[col_name] = label_encoder

    return self

def fit_transform(self, X):
    """
    Fits label encoders for each object/category column and transforms
    non-missing values.

    Parameters:
    df (pd.DataFrame): Input DataFrame with categorical columns.

    Returns:
    pd.DataFrame: Encoded DataFrame with non-missing values transformed.
    """
    df1 = X.copy()
    for col_name in df1.select_dtypes(include=['object',
'category'])).columns:
        series = df1[col_name]
        label_encoder = LabelEncoder()

        # Fit only on non-null values and transform them
        df1[col_name] = pd.Series(
            label_encoder.fit_transform(series[series.notnull()]),
            index=series[series.notnull()].index
        )

        # Store the label encoder for each column
        self.encoders[col_name] = label_encoder

    return df1

def inverse_transform(self, df):
    """
    Decodes non-missing encoded values back to original labels.

    Parameters:
    df (pd.DataFrame): DataFrame with encoded values.

    Returns:
    pd.DataFrame: Decoded DataFrame with original labels.
    """
    df1 = df.copy()
    for col_name in self.encoders:
        enc = self.encoders[col_name]

        # Create a mapping from encoded values back to the original
values
        nmap = dict(zip(enc.transform(enc.classes_), enc.classes_))

        # Apply the mapping only on non-null values
        df1[col_name] = df1[col_name].apply(lambda x: x if np.isnan(x)
else nmap[x])

    return df1

```

```

"""
IgnoreNaNLabelEncoder:
Атрибути:

encoders: Словник для зберігання LabelEncoder для кожного стовпця.
Методи:

fit_transform(df): Цей метод обробляє вхідний DataFrame df. Для кожного
стовпця типу object або category він навчає LabelEncoder лише на ненульових
значеннях, а потім трансформує їх, залишаючи значення NaN незмінними.

inverse_transform(df): Цей метод виконує декодування, перетворюючи коди назад
у їхні оригінальні категоріальні мітки.

```

Використання:

1. Спочатку створюєте екземпляр класу IgnoreNaNLabelEncoder.
 2. Викликаєте fit_transform для вашого DataFrame, щоб виконати кодування міток; метод повертає закодований DataFrame.
 3. Коли вам потрібно повернути зміни, ви викликаєте inverse_transform, щоб декодувати значення назад у початкові мітки.
- Цей підхід гарантує, що пропущені значення зберігаються як під час кодування, так і під час декодування.

Приклад використання:

```

from sklearn.preprocessing import IgnoreNaNLabelEncoder
encoder = IgnoreNaNLabelEncoder()

df = pd.DataFrame({'col1': ['a', 'b', np.nan, 'a', 'b'],
                  'col2': ['cat', 'dog', 'cat', np.nan, 'dog']})
encoder = IgnoreNaNLabelEncoder()
encoded_df = encoder.fit_transform(df)
print("Закодований DataFrame:\n", encoded_df)
decoded_df = encoder.inverse_transform(encoded_df)
print("Декодований DataFrame:\n", decoded_df)

```

```

"""

```

```

import pandas as pd
import numpy as np

```

```

class IgnoreNaNFrequentEncoder:

```

```

    def __init__(self, ascending=False):
        self.encoders = {}
        self.ascending = ascending

```

```

    def fit_transform(self, df):

```

```

        """

```

Кодує значення на основі частотності (не включаючи відсутні значення).

Parameters:

df (pd.DataFrame): Вхідний DataFrame з категоріальними колонками.

Returns:

pd.DataFrame: Закодований DataFrame.

```

        """

```

```

        df1 = df.copy()
        for col_name in df1.select_dtypes(include=['object',
'category']).columns:
            d =
df1[col_name].value_counts().sort_values(ascending=self.ascending).to_dict()
            k = 0

```

```

        for key in d:
            d[key] = k
            k += 1
        self.encoders[col_name] = d
        # Застосовуємо кодування тільки до non-null значень
        dfl[col_name] = dfl[col_name].apply(lambda x: x if pd.isnull(x)
else d[x])

    return dfl

def inverse_transform(self, df):
    """
    Декодує значення, закодовані на основі частотності.

    Parameters:
    df (pd.DataFrame): Вхідний DataFrame з закодованими значеннями.

    Returns:
    pd.DataFrame: Відновлений DataFrame з оригінальними значеннями.
    """
    dfl = df.copy()
    for col_name in self.encoders:
        d = self.encoders[col_name]
        nd = {v: k for k, v in d.items()}
        dfl[col_name] = dfl[col_name].apply(lambda x: x if pd.isnull(x)
else nd[x])

    return dfl

"""
Пояснення:
fit_transform(): Кодує значення в категоріальних стовпцях на основі
частотності, ігноруючи відсутні значення (NaN).
inverse_transform(): Декодує значення назад до їх початкових категоріальних
форм.
Структура пакету zo_ignore_nan_encoders:
zo_ignore_nan_encoders/
├── __init__.py                # Ініціалізація пакету
├── zo_label_encoder.py        # Клас IgnoreNaNLabelEncoder (для
класичного кодування)
└── zo_label_frequent_encoder.py # Клас IgnoreNaNFrequentEncoder (для
кодування на основі частотності)

Приклад використання класу:
import pandas as pd
from zo_ignore_nan_encoders import IgnoreNaNFrequentEncoder
# Приклад даних
df = pd.DataFrame({
    'col1': ['a', 'b', 'a', np.nan, 'c'],
    'col2': ['cat', 'dog', 'cat', 'dog', np.nan]
})
# Використання енкодера на основі частотності
encoder = IgnoreNaNFrequentEncoder()
encoded_df = encoder.fit_transform(df)
print("Закодований DataFrame:\n", encoded_df)
# Відновлення оригінальних значень
decoded_df = encoder.inverse_transform(encoded_df)
print("Декодований DataFrame:\n", decoded_df)

"""

```

Акт впровадження результатів роботи

ПОГОДЖЕНО

Проректор з наукової роботи
Дніпровського національного університету
імені Олеся Гончара

Олег МАРЕНКОВ

« 18 » 05 2026 р.

ЗАТВЕРДЖЕНО

Б.о. першого проректора
Дніпровського національного університету
імені Олеся Гончара
Валентина СІПЧ-БАЛГАБАЄВА

« 18 » 05 2026 р.

АКТ

впровадження результатів роботи **Земляного Олексія Дмитровича**, поданої на здобуття наукового ступеня доктора філософії, на тему
«Розроблення методів та програмного забезпечення інтелектуального імпутування пропусків даних»
 в освітній процес Дніпровського національного університету імені Олеся Гончара

1. Вчена рада факультету прикладної математики та інформаційних технологій у складі 17 осіб заслухала повідомлення аспіранта кафедри інженерії програмного забезпечення та інформаційних технологій Земляного Олексія Дмитровича про результати наукового дослідження та їхнє використання в освітньому процесі галузі інформаційних технологій.

2. Стисла характеристика дослідження

Олексій Земляний досліджує проблему обробки неповних даних, яка впливає на точність аналітичних моделей у задачах класифікації та прогнозування. Автор пропонує нові методи імпутування, такі як UnifiedClassRegrImputer, EntropyImputer, HybridRegrEntropyImputer та CorrelationImputer, які враховують структуру даних, патерни пропусків і залежності між ознаками. Також розроблено алгоритми для перетворення якісних ознак на кількісні зі збереженням інформації про пропуски.

Методи реалізовано згідно з архітектурними принципами бібліотеки scikit-learn на мові програмування Python, що забезпечує їхню інтеграцію в стандартні процеси підготовки даних. Експерименти на медичних та гідрологічних даних підтвердили ефективність запропонованих підходів порівняно з існуючими методами. Робота має практичне значення для покращення якості аналізу даних у різних галузях.

3. Використання в освітньому процесі

Результати дисертаційних досліджень впроваджено в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій факультету прикладної математики та інформаційних технологій ДНУ під час викладання вибіркової освітньої компоненти «Аналіз даних на мові Python» та «Оптимізація та підвищення продуктивності програмного коду» для здобувачів першого (бакалаврського) рівня вищої освіти галузі інформаційних технологій. Окремі теоретичні нароби та результати використано при викладанні дисциплін «Аналіз та візуалізація даних» та «Технології пошуку структури в даних» для здобувачів першого (бакалаврського) рівня вищої освіти та виконанні кваліфікаційних робіт студентами факультету прикладної математики та інформаційних технологій.

4. Відомості про впроваджені об'єкти інтелектуальної власності

4.1. Земляний О.Д., Байбуз О.Г. Огляд методів інтелектуального аналізу даних та методів машинного навчання при прогнозуванні ішемічної хвороби серця // Актуальні проблеми автоматизації та інформаційних технологій. – Дніпро: ДНУ, 2023. – Т.27. – С. 109 – 129. DOI: <http://dx.doi.org/10.15421/432311> [Фахове видання України категорії Б]

(особистий внесок: провів аналіз літературних джерел з теми застосування методів машинного навчання для діагностики ішемічної хвороби серця, систематизував існуючі підходи та алгоритми, а також підготував порівняльну характеристику наборів даних, що використовуються в дослідженнях. Виділив ключові алгоритми та інструменти, оцінив їхню ефективність за показниками точності, чутливості та специфічності.)

4.2. Земляний О.Д., Байбуз О.Г. Методи імпутовання пропусків у даних про ішемічну хворобу серця // Системні технології. Регіональний міжвузівський збірник наукових праць. – Випуск 2(151). – Дніпро, 2024. – С.33 – 49. DOI: <https://doi.org/10.34185/1562-9945-2-151-2024-04> [Фахове видання України категорії Б]

(особистий внесок: розробив та реалізував алгоритми імпутовання пропусків у медичних даних, зокрема адаптував методи на основі класифікації та регресії для роботи з категоріальними та кількісними ознаками. Провів експериментальне тестування запропонованих методів на двох популярних датасетах, оцінив їхню точність та швидкість, проаналізував вплив імпутовання на якість моделей класифікації. Особистий внесок включає програмування алгоритмів, обробку даних та візуалізацію результатів.)

4.3. Земляний О.Д., Байбуз О.Г. Алгоритми імпутовання пропусків у даних на основі ентропії // Системні технології. Регіональний міжвузівський збірник наукових праць. – Випуск 6(155). – Дніпро, 2024. – С.133 – 149. DOI: <https://doi.org/10.34185/1562-9945-6-155-2024-12> [Фахове видання України категорії Б]

(особистий внесок: розробив алгоритм EntropyImputer на основі ентропійного підходу до імпутовання пропусків, який мінімізує умовну ентропію ознак для покращення точності класифікації. Реалізував метод мовою Python, оптимізував код для підвищення продуктивності (зокрема, за допомогою векторизації обчислень) та провів порівняльний аналіз ефективності алгоритму порівняно з традиційними методами. Підготував тестові набори даних та інтерпретував результати експериментів.)

4.4. Земляний О.Д., Байбуз О.Г. Імпутовання пропусків у даних гідрологічного моніторингу // Актуальні проблеми автоматизації та інформаційних технологій. – Дніпро: ДНУ, 2024. – Т.28. – С. 147 – 160. DOI: <http://dx.doi.org/10.15421/432413> [Фахове видання України категорії Б]

(особистий внесок: дослідив особливості гідрологічних даних (зокрема, сезонність та кореляційні зв'язки між показниками) і розробив методи імпутовання, адаптовані для часових рядів. Він провів статистичний аналіз даних гідромоніторингу басейну ріки Дніпро, запропонував підходи до заповнення пропусків на основі кореляційного та регресійного аналізу, а також реалізував алгоритм CorrelationImputer. Особистий внесок включає обробку даних, розробку програмного забезпечення та валідацію результатів на реальних датасетах.)

5. Пропозиції ради

Запропоновано впровадити результати дисертаційної роботи Земляного Олексія Дмитровича на тему «Розроблення методів та програмного забезпечення інтелектуального імпутовання пропусків даних» в освітній процес Дніпровського національного університету імені Олеся Гончара.

Голова вченої ради
факультету прикладної математики
та інформаційних технологій

Секретарка

Олена КІСЕЛЬОВА

Наталія ЛИСИЦЯ