Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Студента Жучкова Сергія Ігоровича
академічної групи 124-19-2
спеціальності 124 Системний аналіз

на тему: «Дослідження поведінки споживача за допомогою методів машинного навчання»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	*Д.ф.-м.н., проф. Купенко О.П.*			
розділів:				
Інформаційно-аналітичний	*Д.ф.-м.н., проф. Купенко О.П.*			
Спеціальний розділ	*Д.ф.-м.н., проф. Купенко О.П.*			
Рецензент				
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище,ініціали)

«____» _____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

студенту *Жучкову С.І.* **академічної групи** *124- 19-2*
спеціальності: *124 Системний аналіз*
на тему «*Методи класифікації незбалансованих наборів даних у задачах Fraud Detection»*
затверджену наказом ректора НТУ «Дніпровська політехніка»
від *16.05.2023 р. №350-с*

Розділ	Зміст	Терміни виконання
1. Інформаційно-аналітичний розділ	*Визначити предметну область дослідження та проблему, що розв'язується. Обґрунтувати методи виконання поставлених завдань.*	24.03.2023 – 07.05.2023
2.Спеціальний розділ	*Розв'язати поставлені задачі: описати структуру об'єкта досліджень, застосувати методи класифікації та провести експерименти для підвищення якості моделей.*	08.05.2023 – 10.06.2023

Завдання видано _____ *Д.ф.-м.н., проф. Купенко О.П.*
(підпис) (прізвище, ініціали)

Дата видачі: 24.01.2023
Дата подання до екзаменаційної комісії: 15.06.2023 р.
Прийнято до виконання _____ Жучков С.І.
(підпис студента) (прізвище, ініціали)

Реферат

Пояснювальна записка: 130 с., 51 рис., 8 табл., 3 додатка, 16 джерел.

Ключові слова: модель, класифікація, незбалансований датасет, методи балансування даних, кластеризація, шахрайство, якість, кластер, кластеризація, прогнозування.

Об'єктом дослідження в цій дипломній роботі є процес прогнозування шахрайських транзакцій, де основним аспектом є вплив незбалансованості класів у наборах даних на якість моделей прогнозування.

Предметом дослідження є методи класифікації незбалансованих даних, а саме їх теоретичні аспекти та практичне застосування на реально існуючих даних і оцінка їх ефективності.

Метою цієї дипломної роботи є дослідження, застосування та порівняння методів класифікації незбалансованих наборів даних в задачах виявлення фроду, також знаходження підходящих методів кластерізацій для покращення якості моделі прогнозування.

Результати дипломної роботи включають детальний аналіз та оцінку методів класифікації незбалансованих даних, включаючи їх ефективність прогнозування шахрайських транзакцій.

Використовується мова програмування Python із бібліотеками, такими як Pandas, NumPy, Matplotlib, Seaborn та Scikit-learn, для обробки даних, виконання аналізу та прогнозування. Інноваційність роботи відрізняється своїм підходом до вибору та застосування методів, які оптимально підходять для обробки незбалансованих даних.

Ця робота вписується у ширший контекст досліджень у сфері машинного навчання та аналізу даних, зокрема у дослідження, що стосуються класифікації незбалансованих даних.

ЗМІСТ

ВСТУП

Останніми роками фірми, які надають фінансові послуги, звертаються до технологій машинного навчання, щоб допомогти виявити та запобігти шахрайству. Ця технологія, яка використовує алгоритми для виявлення шаблонів у великих наборах даних, є ефективним інструментом для виявлення та запобігання шахрайству.

Галузь фінансових послуг особливо вразлива для шахраїв, оскільки вони мають доступ до величезних обсягів даних клієнтів. У результаті традиційних методів виявлення шахрайства часто недостатньо для захисту клієнтів та їхніх коштів. Машинне навчання швидко стає важливим інструментом для запобігання шахрайству та захисту активів клієнтів.

Алгоритми машинного навчання можуть виявляти шаблони в даних, які люди можуть не бачити. Це полегшує ідентифікацію підозрілої активності, наприклад підозріло великих сум транзакцій або кількох транзакцій з одного облікового запису.

Аналізуючи дані клієнтів у режимі реального часу, банки можуть швидко виявляти та запобігати шахрайству. Це допомагає захистити клієнтів від втрат і зберігає безпеку фінансової системи.

Мета дипломної роботи полягає у застосуванні та порівнянні методів класифікації незбалансованих наборів даних в задачах виявлення фроду.

Предметом дослідження є методи класифікації незбалансованих даних, а саме їх теоретичні аспекти та практичне застосування на реально існуючих даних і оцінка їх ефективності.

Об'єктом дослідження є методи класифікації незбалансованих наборів даних в задачах виявлення фроду.

Для досягнення мети дослідження потрібно використати наступні методи:

Аналіз теоретичних аспектів класифікації даних, зокрема методів роботи з незбалансованими даними.

Огляд різноманітних методів класифікації незбалансованих наборів даних, включаючи методи, засновані на деревах рішень, нейронних мережах, ансамблевих методах та інших.

Опис та порівняння кожного з розглянутих методів класифікації.

Розробка експериментальної частини, в рамках якої проведено експеримент з використанням методів класифікації на даних з задачі виявлення фроду.

Аналіз результатів експерименту та формулювання висновків щодо ефективності кожного методу.

ІНФОРМАЦІЙНО-АНАЛІТИЧНІЙ РОЗДІЛ

1.1 Дослідження банківських транзакцій та особливості датасету

Дослідження банківських транзакцій є важливою складовою фінансового аналізу. У зв'язку зі зростанням обсягів банківських транзакцій, з'явилася потреба в розробці ефективних методів аналізу та виявлення аномальних транзакцій, які можуть вказувати на шахрайську діяльність. Такі датасети зазвичай містять інформацію про час, коли була проведена транзакція, типу транзакції, сумі операції в національній валюті, ім’я клієнта, що почав операцію, початковий баланс до операції, баланс клієнта та одержувача після транзакції, ідентифікатор одержувача транзакції та чи є транзакція шахрайством або ні.

Одразу можна зрозуміти головну особливість таких датасетів – категоріальні дані (тип та ідентифікатори клієнта та одержувача). Так як моделі класифікації можуть вчитися лише на числових даних, ці ознаки, якщо з ними не розібратися, ускладнять роботу з даними для цих моделей, тому ці ознаки потребують методів кодування категоріальних ознак, такі як Label Encoding або One-Hot Encoding.

Для успішного та ефективного розуміння даних банківських транзакцій потрібно провести EDA (розвідковий аналіз даних, англ. exploratory data analysis) та залучити релевантні методи кластерізації.

Exploratory Data Analysis - це процес дослідження даних з метою зрозуміти їх структуру, виявити закономірності та корисну інформацію. Він складається з наступних етапів:

1. Первинний аналіз даних: ознайомлення з даними, їх структурою та типами, виявлення пропущених значень, викинутих стовпців і рядків, статистичний аналіз розподілу даних.

2. Візуалізація даних: відображення даних у вигляді графіків, діаграм, хмарних графіків, гістограм, що дозволяє отримати зображення розподілу даних та співвідношення між ними.

3. Розуміння залежностей: пошук залежностей між різними змінними за допомогою кореляційних матриць, теплових карт, факторного аналізу та інших методів.

4. Обробка даних: виявлення аномальних даних, опрацювання пропущених значень, вибір змінних для подальшого аналізу та побудови моделей.

5. Перевірка гіпотез: тестування гіпотез щодо залежності між змінними, відповідності даних певному розподілу, перевірка статистичних гіпотез.

6. Висновки: на основі проведеного аналізу формулювання висновків та рекомендацій щодо подальшого використання даних, побудови моделей або виконання подальших досліджень.

EDA є важливим етапом у будь-якому проекті з аналізу даних, оскільки він дозволяє зрозуміти характеристики та особливості даних, з якими ми маємо справу. Це може допомогти визначити, які моделі, алгоритми та методи аналізу можуть бути найбільш ефективними для нашої задачі, а також дозволяє виявити потенційні проблеми та виклики, пов'язані з даними.

1.2 Методи підвищення якості прогнозу моделей класифікації

Методи підвищення точності моделей класифікації включають різні техніки, які можуть бути використані для покращення якості прогнозів моделі.

Розглянемо деякі з них:

1. Оптимізація даних:

Очищення та підготовка даних є важливим етапом у будь-якому проекті машинного навчання. Неочищені або погано підготовлені дані можуть значно вплинути на якість моделі та її точність.

На етапі очищення даних виконують перевірку на наявність відсутніх значень та їх обробку (видалення рядків з відсутніми значеннями, заповнення пропущених даних тощо) та видалення дублікатів.

Видалення викидів:

Викиди (англ. outliers) - це значення, що виділяється із звичайного діапазону даних і може бути помилкою вимірювання або відображенням особливостей вихідних даних. Наявність викиди у даних може негативно вплинути на точність моделі машинного навчання, оскільки модель може зосередитись на неправильних залежностях і, отже, недооцінити або переоцінити результат.

Існують різні методи обробки викидів, зокрема, видалення або заміна викидів. У випадку видалення викидів, можна використовувати статистичні методи для визначення діапазону "нормальних" значень, а потім видалити ті значення, що виходять за межі цього діапазону. Інший підхід - заміна значень, що виходять за межі діапазону, на інші значення, наприклад, на середнє або медіану даних.

Однак, не завжди видалення або заміна викидів є корисним підходом. Наприклад, у деяких випадках, викид може бути корисним сигналом для моделі, особливо, якщо він представляє реальну аномалію, яка може мати значення для вирішення задачі. Тому перед застосуванням методів обробки викидів, необхідно ретельно проаналізувати дані та зважити на вплив викидів на точність моделі.

Encoding або кодування категоріальних даних:

Кодування категоріальних даних - це процес перетворення категоріальних змінних, які приймають обмежену кількість можливих значень, на числові значення, які можуть бути використані в моделях машинного навчання. Це необхідно, оскільки більшість алгоритмів машинного навчання не можуть працювати з категоріальними змінними у вигляді текстових міток.

Існує кілька способів кодування категоріальних даних, включаючи:

- Бінарне кодування (One-Hot Encoding): кожне можливе значення категорії представлено як бінарний вектор. Наприклад, якщо ми маємо змінну "кольори" з трьома можливими значеннями - червоний, зелений та синій, то ми можемо створити три нові змінні "колір_червоний", "колір_зелений" та "колір_синій", і призначити їм значення 1 або 0, в залежності від того, який колір є для кожного запису. Цей метод використовується для збереження всієї інформації про категорію, але призводить до збільшення кількості ознак у даних.

- Частотне кодування (Frequency Encoding): кожне значення категорії замінюється його частотою в даних. Наприклад, якщо ми маємо змінну "кольори" з трьома можливими значеннями - червоний, зелений та синій, то ми можемо замінити кожен колір на його частоту, тобто кількість разів, коли він зустрічається в даних. Цей метод простіший, ніж бінарне кодування, але може викликати проблеми, якщо значення категорії мають однакову частоту, оскільки вони будуть мати однакове закодоване значення.

- Label Encoding: кожна категорія замінюється на числове значення, яке відповідає її позиції в послідовності унікальних значень. Наприклад, якщо ми маємо змінну "кольори" з трьома можливими значеннями - червоний, зелений та синій, то ми можемо замінити кожен колір на числа 1, 2 та 3 відповідно. Цей метод використовується, коли порядок категорій не має значення, але може призвести до помилкового розуміння, що значення категорій мають певний порядок.

- Target Encoding: кожне значення категорії замінюється на середнє значення цільової змінної для цієї категорії. Наприклад, якщо ми маємо змінну "місто" та цільову змінну "ціна", то ми можемо замінити кожне місто на середню ціну в цьому місті. Цей метод використовується, коли категорії можуть мати

різний вплив на цільову змінну, але може призвести до перенавчання моделі, якщо певні категорії мають надмірний вплив на цільову змінну.

Масштабування або нормалізація числових змінних:

Масштабування або нормалізація числових змінних - це процес приведення значень числових змінних до певного діапазону або розподілу. Це робиться для того, щоб зробити числові змінні порівнянними, а також для підвищення швидкості та якості навчання моделі.

Масштабування може здійснюватися за допомогою двох підходів: нормалізації та стандартизації. Нормалізація зазвичай використовується, коли діапазон значень змінних різний, а стандартизація - коли змінні мають різні дисперсії.

При нормалізації значення змінної масштабуються до діапазону від 0 до 1 або від -1 до 1. Це робиться шляхом віднімання мінімального значення змінної та поділу на різницю між максимальним та мінімальним значеннями змінної.

При стандартизації значення змінної масштабуються так, щоб вони мали середнє значення 0 та стандартне відхилення 1. Це робиться шляхом віднімання середнього значення змінної та поділу на стандартне відхилення.

Масштабування змінних дозволяє моделі краще узгоджуватися з даними та підвищує її точність та стійкість.

2. Feature selection або вибор ознак:

Відбір ознак - це процес вибору підмножини ознак з вхідних даних, які є найбільш інформативними для побудови моделі машинного навчання. Це може бути корисним для зменшення кількості ознак, що використовуються в моделі, що дозволяє зменшити ризик перенавчання, збільшити швидкість навчання та покращити загальну ефективність моделі.

Існує багато методів відбору ознак, такі як:

- Filter methods (методи фільтрації): використовують статистичні метрики для відбору ознак, які мають найбільший вплив на вихідну змінну. Приклади таких метрик - кореляційний коефіцієнт Пірсона, mutual information, chi-squared test тощо.

- Wrapper methods (методи упаковки): використовують ітеративний підхід, де модель машинного навчання навчається на різних підмножинах ознак і відбирається найкраща підмножина на основі валідаційних метрик. Приклади таких методів - Recursive Feature Elimination, Forward Selection, Backward Elimination тощо.
- Embedded methods (вбудовані методи): відбір ознак вбудований у процес навчання моделі машинного навчання. Найбільш відомі приклади вбудованих методів - Lasso Regression, Elastic Net, Ridge Regression тощо.

3. Cross-validation або кросс-валідація:

Кросс-валідація - це метод оцінки якості моделі машинного навчання, який дозволяє оцінити, наскільки добре модель буде працювати на нових невідомих даних. Зазвичай кросс-валідація використовується для оцінки якості моделі перед її застосуванням на реальних даних.

Ідея полягає в тому, що наявний набір даних розділяється на кілька частин, наприклад, 5 частин (5-fold cross validation) або 10 частин (10-fold cross validation). Далі модель навчається на 4 частинах, а потім перевіряється на залишковій 1 частині. Ця процедура повторюється кілька разів, при цьому кожна з частин використовується як тестовий набір даних, а решта - як тренувальний.

Після цього обчислюється середнє значення метрики якості (наприклад, точності або F1-мери) для всіх тестових наборів даних. Ця метрика дозволяє оцінити якість моделі, тобто наскільки добре вона здатна передбачати класи для нових невідомих даних.

Використання кросс-валідації дозволяє уникнути перенавчання моделі на конкретних даних і забезпечити більш точну оцінку її роботи на нових даних.

4. Оптимізація гіперпараметрів моделі:

Оптимізація гіперпараметрів моделі - це процес знаходження оптимальних значень гіперпараметрів моделі, які забезпечують кращу продуктивність моделі на тестових даних.

Зазвичай, для оптимізації гіперпараметрів моделі використовують методи пошуку гіперпараметрів, такі як решітчатий пошук, випадковий пошук або оптимізація градієнтного спуску. У решітчатому пошуку використовуються певні значення гіперпараметрів, а потім обчислюється метрика якості моделі для кожної комбінації значень гіперпараметрів. Випадковий пошук використовує випадкові значення гіперпараметрів для обчислення метрики якості моделі. Оптимізація

градієнтного спуску використовує алгоритм градієнтного спуску для знаходження оптимальних значень гіперпараметрів.

Зазвичай, оптимізація гіперпараметрів моделі відбувається шляхом поділу даних на навчальний та тестовий набори, а потім знаходження оптимальних значень гіперпараметрів моделі на навчальному наборі даних з використанням методів пошуку гіперпараметрів. Після знаходження оптимальних значень гіперпараметрів моделі на навчальному наборі даних, модель тестується на тестовому наборі даних, щоб перевірити її продуктивність.

5. Ансамблеві моделі машинного навчання:

Ансамбль моделей - це підхід до побудови моделі, який поєднує кілька індивідуальних моделей з метою отримання більш точних та надійних прогнозів. Замість використання окремої моделі, ансамбль об'єднує прогнози кількох моделей та робить узагальнений прогноз на основі їх результатів.

Існує кілька типів ансамблів моделей, найпоширеніші з яких такі:

- Беггінг (Bagging): використовується для зменшення варіації моделі шляхом тренування кількох моделей на різних підмножинах даних. Кожна модель незалежно тренується, а результати їх прогнозів комбінуються, наприклад, шляхом голосування або усереднення.

- Випадковий ліс (Random Forest): це тип ансамблю зачеплених моделей, де використовується рішучий ліс як базова модель. Рішучий ліс використовує дерева рішень для прогнозування і комбінує результати декількох дерев для отримання кінцевого прогнозу.

- Послідовний стекінг (Sequential Stacking): використовується для комбінування результатів кількох моделей, де вихідні дані однієї моделі використовуються як вхідні дані для іншої моделі. Результати прогнозів кожної моделі комбінуються для отримання кінцевого прогнозу.

- Бустінг (Boosting): це метод, в якому моделі будуються послідовно, при цьому кожна наступна модель спрямовується на коригування помилок попередніх моделей. Бустінг створює сильну модель шляхом комбінування слабких моделей.

Ансамбль моделей може бути підібраний та налаштований для досягнення кращих результатів. Однак, використання ансамблювання може призвести до

більш складної моделі, що вимагає більше обчислювальних ресурсів та часу на навчання.

1.3 Нерівномірний розподіл класів та проблема нерівномірної класифікації

Нерівномірність розподілу класів - це ситуація, коли дані, які необхідно класифікувати, мають нерівномірний розподіл по класам. Якщо один клас має набагато більше прикладів, ніж інший, то модель буде схильною до прийняття рішень на користь більшого класу. Це може призвести до невдалих прогнозів для меншої класу. Проблема нерівномірної класифікації полягає у тому, що модель не може адекватно виконувати класифікацію меншості, і як результат, може допускати помилки в її класифікації.

Для вирішення проблеми нерівномірної класифікації можна використовувати різні методи, такі як:

- Збільшення вибірки меншої класу (oversampling): цей метод полягає в копіюванні прикладів меншої класу з метою збільшення її розміру до рівня більшої класу. Це дозволяє зменшити проблему нерівномірності розподілу класів, але може призвести до перенавчання моделі.
- Зменшення вибірки більшої класу (undersampling): цей метод полягає у випадковому видаленні прикладів більшої класу до досягнення рівномірного розподілу класів. Це також може призвести до перенавчання моделі.

- Використання ваг для класів (class weighting): цей метод полягає в призначенні ваг класам в залежності від їх розміру. Наприклад, меншому класу можна призначити більшу вагу, щоб зменшити ймовірність помилки класифікації цього класу. Цей метод дозволяє зменшити проблему нерівномірності розподілу класів, не призводячи до перенавчання моделі.

- Використання алгоритмів з урахуванням нерівномірної класифікації (class-imbalanced algorithms): деякі алгоритми машинного навчання, такі як XGBoost, LightGBM та CatBoost, мають вбудовані методи для вирішення проблеми нерівномірної класифікації. Ці алгоритми дозволяють встановлювати різні параметри, такі як scale_pos_weight, який дозволяє налаштувати ваги класів.

- При вирішенні проблеми нерівномірної класифікації важливо вибрати метод, який дозволяє зменшити нерівномірність розподілу класів, не призводячи до перенавчання моделі і забезпечує якісну класифікацію для всіх класів.

1.4 Методи балансування даних

Методи балансування даних є важливими для вирішення проблеми нерівномірного розподілу класів у наборі даних. Коли класи у наборі даних представлені нерівномірно, це може призвести до поганої узгодженості моделі та неправильних прогнозів.

Основна причина, чому потрібно застосовувати методи балансування даних, полягає в тому, що моделі машинного навчання мають тенденцію віддавати перевагу класу з більшою кількістю прикладів, вважаючи його більш представником та важливим. Це може призвести до неправильної класифікації меншої кількості класу або ігнорування його зовсім.

Методи балансування даних допомагають уникнути цих проблем та покращують якість моделі класифікації. Вони забезпечують більш справедливу репрезентацію обох класів, зменшуючи перекос і покращуючи здатність моделі розрізняти та класифікувати обидва класи.

У цьому підрозділі ми ознайомимось з ними детальніще:

1. Метод Oversampling є одним з підходів до балансування даних, що полягає у створенні додаткових прикладів з меншої кількості класу, щоб збалансувати нерівномірність розподілу класів у наборі даних.

Один з підходів до збільшення кількості записів - це дублювання вже наявних записів. Однак цей метод може призвести до перенавчання моделі, оскільки він додає ті самі записи знову та знову, збільшуючи ймовірність переобучення.

Інший підхід до oversampling полягає в застосуванні методу SMOTE, що створює нові записи, шляхом генерації синтетичних прикладів. SMOTE вибирає випадкові записи з меншої кількості класу і використовує їх, щоб створити синтетичні записи, додавши до них випадкові значення між оригінальними записами. SMOTE дозволяє створити нові приклади, що допомагають уникнути проблем перенавчання та збільшують різноманітність даних.

Алгоритм SMOTE полягає в наступних кроках:

1. Вибір випадкового запису з меншістю класу.

2. Визначення його k найближчих сусідів в цьому ж класі за допомогою метрики відстані, наприклад, Евклідової відстані.

3. Вибір випадкового запису з цих k сусідів.

4. Генерація нового запису попередньо не відомого класу на відрізку, який з'єднує початковий запис та випадкового сусіда з меншості класу, використовуючи формулу:

new_record = original_record + (random_number * (neighbor_record - original_record))

де random_number - випадкове число від 0 до 1, neighbor_record - запис в меншій кількості класу, вибраний на кроці 3.

5. Повторення кроків 1-4 задану кількість разів, щоб отримати більшу кількість штучних записів.

Для кращого розуміння цього методу візуалізуємого його роботу над даними випадкового датасету:

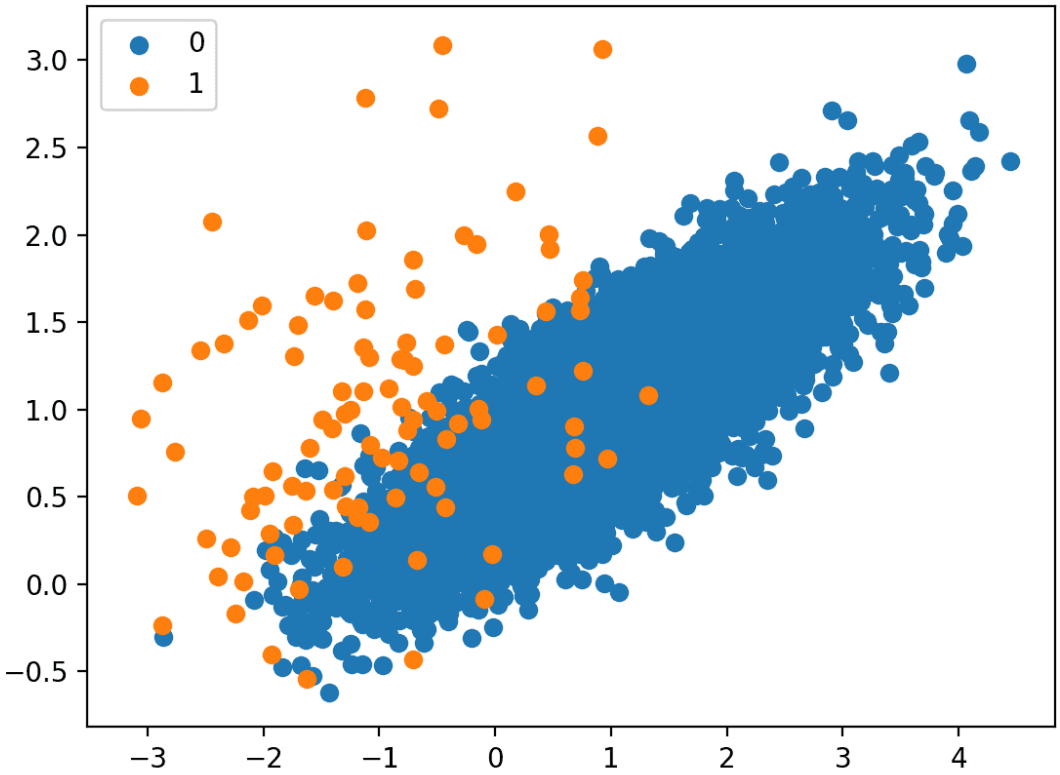

Рисунок 1.1 Графік розділення класу більшості(синім) та клас меншості(помаранчевий)

Ця точкова діаграма набору даних показує велику масу точок, які належать до класу більшості (синій), і невелику кількість точок, розподілених для класу меншості (помаранчевий). Ми бачимо певну міру перекриття між двома класами.

Ми маємо перед собою незбалансований набір, тому за допомогою методу SMOTE створимо синтетичні записи, додавши до них випадкові значення між оригінальними записами.

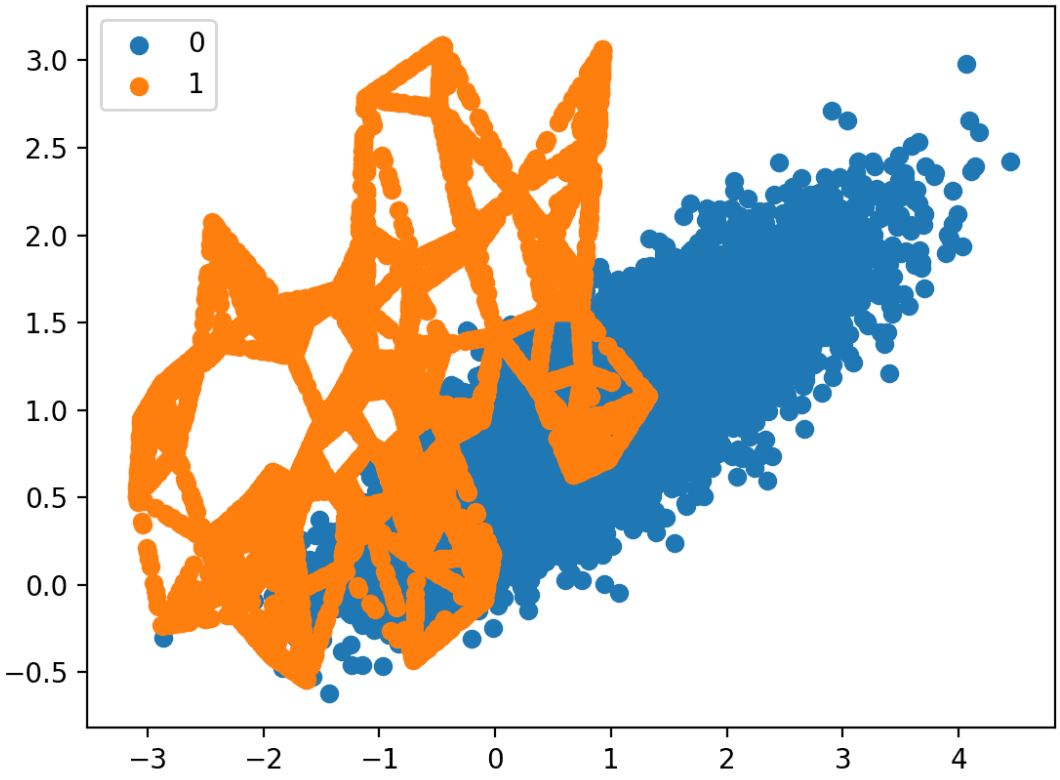

Рисунок 1.2 Результат методу SMOTE

Як можемо побачити, метод згенерував нові записи та доповнив їх до класу меньшості та тепер графік показує багато інших прикладів у класі меншості, створених уздовж ліній між вихідними прикладами в класі меншості.

Основною перевагою SMOTE є те, що він допомагає уникнути перенавчання моделі, що може статися при повторному використанні наявних записів, що належать до меншості класу. Використання синтетичних записів дозволяє моделі навчитися різним характеристикам меншості класу та забезпечує більш рівномірне розподіл класів.

Однак, SMOTE також має свої недоліки. Наприклад, якщо дані складаються з багатьох дуже вузьких кластерів, то SMOTE може призвести до створення більш широких кластерів та знизити точність моделі. Крім того, SMOTE може не підходити для деяких даних, якщо вони мають велику кількість шуму або випадкових даних, що не пов'язані з конкретним класом. Також, важливо пам'ятати, що SMOTE не вирішує проблему зі збалансуванням класів, якщо головною причиною їх нерівномірності є неправильна підготовка даних або поганий вибір ознак.

Таким чином, SMOTE є корисним інструментом для розв'язання проблеми нерівномірного розподілу класів у наборі даних, але він має свої обмеження і вимагає обережного застосування.

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

(* *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* *)
if $N < 100$
  then Randomize the T minority class samples
    $T = (N/100) * T$
    $N = 100$
endif
$N = (int)(N/100)$ (* *The amount of SMOTE is assumed to be in integral multiples of 100.* *)
k = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(* *Compute k nearest neighbors for each minority class sample only.* *)
for $i \leftarrow 1$ **to** T
    Compute k nearest neighbors for i, and save the indices in the $nnarray$
    Populate(N, i, $nnarray$)
endfor

Populate(N, i, $nnarray$) (* *Function to generate the synthetic samples.* *)
while $N \neq 0$
    Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
    for $attr \leftarrow 1$ **to** $numattrs$
        Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][$
        Compute: gap = random number between 0 and 1
        $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
    endfor
    $newindex{+}{+}$
    $N = N - 1$
endwhile
return (* *End of Populate.* *)
End of Pseudo-Code.

Рисунок 1.3 Псевдокод методу SMOTE

2. Інший метод, що базується на SMOTE, - це ADASYN (Adaptive Synthetic Sampling), алгоритм, що збільшує кількості зразків меншості у наборі даних, що дозволяє зменшити нерівномірність розподілу класів та враховує плотність класів у просторі ознак.

ADASYN (Adaptive Synthetic Sampling) - це алгоритм для збільшення кількості зразків меншості у наборі даних, що дозволяє зменшити нерівномірність розподілу класів. Він працює на основі SMOTE (Synthetic Minority Over-sampling Technique), але враховує плотність класів у просторі ознак.

Основна ідея ADASYN полягає в тому, що він генерує штучні зразки меншості шляхом вибору точки з меншості та її ближайших сусідів. Однак замість генерації нового зразка на прямій, що з'єднує вихідну точку і одного з її сусідів, ADASYN генерує зразки в околі цієї прямої залежно від рівня нерівномірності класів.

Конкретніше, для кожної точки меншості ADASYN обчислює вагу, яка відображає, наскільки багато зразків меншості потрібно синтезувати для цієї точки. Вага розраховується як відношення кількості зразків більшості, які знаходяться в околі точки, до загальної кількості зразків меншості в цьому околі. Очевидно, що чим менше прикладів меншості в околі, тим більша вага надається точці меншості.

Після обчислення ваг для кожної точки меншості, алгоритм вибирає кількість зразків, яку потрібно створити, і для кожної точки вибирає ближайші сусіди з більшості. Зразки генеруються в околі лінії, що з'єднує точку меншості і вибраного сусіда з більшості, пропорційно вагам точок меншості.

Однією з переваг ADASYN є те, що він адаптується до нерівномірного розподілу класів у просторі ознак і створює більше штучних зразків в областях, де даних менше, а менше зразків там, де даних більше. Це дозволяє зменшити ризик перенавчання моделі та покращити її загальну здатність до узагальнення.

Алгоритм ADASYN дуже зрозуміло описав Haibo He, Yang Bai та Edwardo A. Garcia у своїй статті [1] про ADASYN :

Input

(1) Training data set D_{tr} with m samples $\{\mathbf{x_i}, y_i\}$, $i = 1, ..., m$, where $\boldsymbol{x}_i$ is an instance in the n dimensional feature space $\boldsymbol{X}$ and $y_i \in Y = \{1, -1\}$ is the class identity label associated with $\boldsymbol{x}_i$. Define m_s and m_l as the number of minority class examples and the number of majority class examples, respectively. Therefore, $m_s \leq m_l$ and $m_s + m_l = m$.

Procedure

(1) Calculate the degree of class imbalance:

$$d = m_s/m_l \tag{1}$$

where $d \in (0,\ 1]$.

(2) If $d < d_{th}$ then (d_{th} is a preset threshold for the maximum tolerated degree of class imbalance ratio):

(a) Calculate the number of synthetic data examples that need to be generated for the minority class:

$$G = (m_l - m_s) \times \beta \tag{2}$$

Where $\beta \in [0,\ 1]$ is a parameter used to specify the desired balance level after generation of the synthetic data. $\beta = 1$ means a fully balanced data set is created after the generalization process.

(b) For each example $\boldsymbol{x_i} \in minorityclass$, find K nearest neighbors based on the Euclidean distance in n dimensional space, and calculate the ratio r_i defined as:

$$r_i = \Delta_i/K, \quad i = 1, ..., m_s \tag{3}$$

where Δ_i is the number of examples in the K nearest neighbors of $\boldsymbol{x}_i$ that belong to the majority class, therefore $r_i \in [0,\ 1]$;

(c) Normalize r_i according to $\hat{r}_i = r_i / \sum_{i=1}^{m_s} r_i$, so that $\hat{r}_i$ is a density distribution ($\sum_i \hat{r}_i = 1$)

(d) Calculate the number of synthetic data examples that need to be generated for each minority example $\boldsymbol{x}_i$:

$$g_i = \hat{r}_i \times G \tag{4}$$

where G is the total number of synthetic data examples that need to be generated for the minority class as defined in Equation (2).

(e) For each minority class data example $\boldsymbol{x}_i$, generate g_i synthetic data examples according to the following steps:

Do the **Loop** from 1 to g_i:

(i) Randomly choose one minority data example, $\boldsymbol{x}_{zi}$, from the K nearest neighbors for data $\boldsymbol{x}_i$.

(ii) Generate the synthetic data example:

$$\boldsymbol{s_i} = \boldsymbol{x_i} + (\boldsymbol{x_{zi}} - \boldsymbol{x_i}) \times \lambda \tag{5}$$

where $(\boldsymbol{x_{zi}} - \boldsymbol{x_i})$ is the difference vector in n dimensional spaces, and λ is a random number: $\lambda \in [0,\ 1]$.

End **Loop**

Рисунок 1.4 Псевдокод методу ADASYN

Алгоритм ADASYN по суті є розширенням методу SMOTE. Він працює подібно до SMOTE, але додатково враховує густоту розподілу прикладів. Для цього алгоритм використовує k-найближчих сусідів та збільшує кількість прикладів шляхом додавання випадкового коефіцієнту до відстані між вибраним прикладом та його найближчим сусідом. Це дозволяє генерувати більше

синтетичних прикладів для меншої класу, які знаходяться в менш щільно замінених областях, а менше для тих, які знаходяться в більш щільно замінених областях. Це покращує ефективність балансування класів та зменшує кількість зайвих синтетичних прикладів.

Незважаючи на те, що ADASYN є одним з найбільш популярних методів боротьби з незбалансованими класами, але є кілька недоліків:

1. Чутливість до шуму: метод ADASYN генерує нові приклади шляхом зміщення інших прикладів уздовж ліній від одного прикладу до іншого. Це може призвести до того, що генеровані приклади будуть більш схожі на шумові або випадкові відхилення від оригінальних прикладів, що може погіршити якість класифікації.

2. Обчислювальна складність: метод ADASYN вимагає обчислення відстаней між кожним з міноритарних прикладів та їхніми K найближчими сусідами, що може бути обчислювально складно для великих даних.

3. Залежність від K: якщо K занадто мале, то метод може бути недостатньо чутливим до структури даних, а якщо K занадто велике, то може виникнути проблема з вибором найближчих сусідів для прикладів в густонаселених областях.

4. Залежність від порогу: якщо поріг максимального ступеня дисбалансу встановлено неправильно, то метод може генерувати занадто багато або занадто мало штучних прикладів, що може призвести до перенавчання або недонавчання моделі.

5. Відсутність гарантії вдосконалення результатів: метод ADASYN не гарантує покращення результатів класифікації та може привести до погіршення результатів у деяких випадках.

Ще одним методом балансування є Class weighting:

Class weighting (вагова налаштування класів) - це метод зменшення впливу нерівномірної розподіленості класів в навчальному наборі на результати моделі. Коли класи нерівномірно розподілені, модель може навчитися краще прогнозувати домінуючий клас, залишаючи меншість класів неохопленою. В такому випадку важко визначити відносну важливість кожного класу, іноді невдалим рішенням є використання простої точності (accuracy) як метрики оцінки результатів.

Метод вагової настройки полягає в наданні більшої ваги меншому класу в процесі навчання. Для цього вводиться параметр ваги (class weight), який може бути присвоєний кожному класу окремо. Параметри ваги можуть бути обчислені автоматично на основі розподілення класів у навчальному наборі або задані вручну.

Під час навчання, класи з вищою вагою будуть мати більший вплив на процес оптимізації моделі, що дозволяє досягти кращих результатів для менших класів. Зокрема, вагова настройка може допомогти уникнути перенавчання (overfitting) на домінуючих класах і покращити точність прогнозування менших класів.

Недоліком методу вагової настройки є те, що він може призвести до зниження точності для домінуючого класу, якщо вага для меншого класу встановлена занадто високо. Також, при використанні вагової настройки необхідно бути обережним і враховувати, що вага класу може впливати на рішення моделі, тому її необхідно встановлювати з урахуванням конкретних вимог до результатів.

1.5 Методи групування даних

Методи групування даних, такі як кластеризація, відіграють важливу роль у різних аспектах аналізу даних і машинного навчання. Основні цілі та застосування методів групування даних включають:

1. Розуміння структури даних: Методи групування даних дозволяють виявити приховану структуру та патерни в наборі даних. Вони допомагають ідентифікувати схожі групи об'єктів або спостережень, що можуть мати подібні характеристики або взаємозв'язки.

2. Візуалізація даних: Кластеризація дозволяє візуалізувати великий набір даних, розмістивши його у вигляді груп або кластерів. Це допомагає сприйняти та зрозуміти структуру даних, а також виявити аномалії або випадки, які виходять за межі груп.

3. Стиснення даних: Кластеризація може використовуватись для стиснення даних, замінюючи групи даних на їх центри або представників. Це може зменшити розмір даних та скоротити обчислювальні витрати при подальшому аналізі.

4. Класифікація та передбачення: Методи групування даних можуть використовуватись як попередній етап для побудови моделей класифікації або передбачення. Вони можуть допомогти виділити групи даних з подібними властивостями, що сприяє покращенню якості моделей.

6. Виявлення аномалій та шахрайства: Методи групування даних можуть виявляти аномалії або викиди в наборі даних, включаючи випадки шахрайства або несправедливої діяльності. Вони допомагають виділити незвичайні зразки, що відрізняються від нормального поведінки.

Тепер розглянемо деякі методи кластеризації більш детальніше:

Кластеризація k-середніх (k-means clustering) - це один з найпоширеніших алгоритмів групування даних. Вона використовується для розділення набору даних на кілька груп або кластерів на основі їх схожості за деякими ознаками. Основна ідея полягає в тому, щоб знайти центри кластерів (k-середні) та призначити кожен об'єкт до найближчого за відстанню центру.

Основні кроки алгоритму k-середніх такі:

1. Вибрати кількість кластерів (k), яку потрібно сформувати з набору даних.

2. Випадковим чином ініціалізувати початкові к-середні (центри кластерів).

3. Повторювати до збіжності або до досягнення максимальної кількості ітерацій:

- Призначити кожен об'єкт до найближчого за відстанню центру кластеру.
- Оновити центри кластерів, обчислюючи середні значення об'єктів у кожному кластері.

4. Вивести остаточний набір кластерів та їх центри.

Кластеризація k-середніх має декілька переваг, зокрема простоту в реалізації, швидкість обчислень і підходить для великих наборів даних. Вона може використовуватись для виявлення прихованих структур та групування схожих об'єктів. Проте, важливо враховувати, що результати кластеризації можуть залежати від початкових значень центрів кластерів і можуть бути чутливі до шуму та випадкових факторів.

Кластеризація k-середніх може бути застосована в різних областях, таких як аналіз даних, комп'ютерний зір, біоінформатика, маркетингові дослідження, виявлення шахрайств та багато інших, де важливо групувати подібні об'єкти для подальшого аналізу та виявлення закономірностей.

Нижче наданий псевдокод алгоритму к-середніх зі статті [5]:

A. K means Algorithm

Input:

D = {d1, d2, ..., dn} //set of n data items.

k // Number of desired cluster

Output:

A set of *k* clusters.

Step:

1. Arbitrarily choose *k* data-items from D as initial centroids;
2. Repeat

   Assign each item *di* to the cluster which has the closest centroid;

   Calculate new mean for each cluster;

   Until convergence criteria is met.

Fig. 1 Pseudocode of k means algorithm

Рисунок 1.5 Псевдо код алгоритму к-середніх

Нечітка кластеризація (fuzzy clustering) є іншим методом групування даних, який використовується для призначення кожного об'єкта набору даних до кластеру з певною ймовірністю членства. У відміну від кластеризації k-середніх, де об'єкт належить лише до одного кластеру з абсолютною впевненістю, в нечіткій кластеризації об'єкти можуть мати градацію впевненості у своєму членстві до різних кластерів.

Основні особливості нечіткої кластеризації:

1. Визначення кількості кластерів (k), аналогічно до кластеризації k-середніх.

2. Визначення функції належності, яка вказує ступінь членства кожного об'єкта до кожного кластеру.

3. Побудова матриці належності, де кожен елемент вказує ступінь членства об'єкта відносно кожного кластеру.

4. Оновлення центрів кластерів з урахуванням ступенів належності об'єктів.

5. Повторення процесу до збіжності або до заданої умови зупинки.

Нечітка кластеризація дозволяє моделювати неоднозначність та невизначеність при групуванні даних, що корисно в задачах, де об'єкти можуть мати часткову приналежність до кількох кластерів одночасно. Цей метод знайшов застосування в багатьох областях, включаючи розпізнавання образів, обробку природних мов, біоінформатику та інші. Важливо враховувати, що нечітка

кластеризація вимагає визначення додаткових параметрів, таких як функція належності, і може бути чутливою до початкових значень.

Нижче наданий псевдокод алгоритму нечіткої кластерізації зі статті [6]:

FCM Algorithm :
Input : θ (k,l), N
Output U^*_{FCM}, V^*_{FCM}
1. *Initialize Partition* $U^{(0)}$ *randomly*
2. *for i = 1 to n*
3. *for k = 1 to c*
4. *Repeat for j = 1, 2, 3, ...*
5. *Update centroid* $V^{(0)}$ *with* $U^{(j-1)}$ *Using (3)*
6. *Compute Distance* $D^{(j)}$ *with* $V^{(j)}$
7. *Update Partition Matrix* $U^{(j)}$ *with* $D^{(j)}$ *using (5)*
8. *Until* $||U^{(j)} - U^{(j-1)}|| < \epsilon$
9. *end*
10. *end*
11. *Return* $U^*_{FCM} \leftarrow U^{(j)}$ *and* $V^*_{FCM} \leftarrow V^{(j)}$

Рисунок 1.6 Псевдокод алгоритму нечіткої кластерізації

1.6 Методи класифікації незбалансованих наборів даних

XGBoost (eXtreme Gradient Boosting) - це алгоритм машинного навчання з відкритим кодом, який використовує градієнтний бустінг для підвищення точності моделей класифікації та регресії. Він є популярним в індустрії та наукових дослідженнях завдяки своїй ефективності та швидкості.

XGBoost є розширенням градієнтного бустінгу, додавши багато нових функцій та покращень, таких як розріджена матриця, підтримка стандартних API для моделей машинного навчання та багато іншого. В основі XGBoost лежить бібліотека на мові програмування C++, яка підтримує паралельну обробку даних на рівні батчів.

Алгоритм XGBoost використовує градієнтний бустінг над деревами рішень. Основна ідея полягає в тому, щоб послідовно побудувати послідовні дерева рішень і використовувати їх для корекції попередніх помилок. Кожне наступне дерево побудоване на основі остач, залишених попереднім деревом.

Алгоритм має декілька параметрів, які можна настроїти для отримання найкращих результатів. Наприклад, кількість дерев, що будуються, глибина дерев, критерії розбиття, розмір батчу та інші.

Основна ідея полягає в тому, щоб послідовно додавати нові дерева до ансамблю, коригуючи попередні прогнози моделі, щоб отримати кращу точність прогнозів.

Основний алгоритм XGBoost можна розбити на наступні кроки:

1. Ініціалізувати модель: визначити стартове прогнозоване значення для всіх прикладів і створити модель, яка буде постійно оновлюватись.

2. Обчислити градієнт та гессіан для всіх прикладів: градієнт і гессіан - це вектори, які вказують на напрямок найшвидшого зростання функції в кожній точці. Градієнт використовується для покращення моделі на кожній ітерації, а гессіан допомагає підвищити ефективність покращення моделі.

3. Побудувати дерево рішень: дерево будується шляхом розділення даних на підмножини, використовуючи критерій інформативності (наприклад, критерій Джині або ентропія). Алгоритм рекурсивно розділяє дані на підмножини, поки не досягне максимальної глибини дерева або не досягне мінімальної кількості елементів в кожному листку.

4. Розрахувати приналежність до кожного листка дерева для кожного прикладу: кожен приклад розподіляється на шляху вниз по дереву, завдяки чому кожен елемент даних потрапляє в листок, до якого належить.

5. Обчислити розбіжність між прогнозованими та фактичними значеннями: це різниця між прогнозованим значенням та фактичним значенням цільової змінної для кожного елемента даних.

6. Оновити модель, додаючи нове дерево до ансамблю: нове дерево додається до моделі, і прогнозовані значення оновлюються з урахуванням нової інформації. Цей процес повторюється доти, доки не досягнута задана кількість дерев або поки досягнута задана точність.

7. Здійснити прогноз: для нових елементів даних здійснюється прогноз за допомогою побудованого ансамблю дерев.

Основними перевагами XGBoost є швидкість та ефективність роботи з великими наборами даних, а також можливість автоматичного вибору гіперпараметрів, що дозволяє покращити точність моделі. Крім того, XGBoost має вбудовану підтримку обробки пропущених даних та можливість роботи з різними типами даних.

Недоліками XGBoost можуть бути:
- складність налаштування параметрів моделі;
- необхідність розуміння технічних аспектів реалізації.

Нижче предоставлено псевдокод зі статті [2]

- Initialize sample weights $w_i^{(0)} = \frac{1}{l}, i = 1, \ldots, l.$
- For all $t = 1, \ldots, T$
  - Train base algo b_t, let ϵ_t be it's training error.
  - $\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$.
  - Update sample weights: $w_i^{(t)} = w_i^{(t-1)} e^{-\alpha_t y_i b_t(x_i)}, i = 1, \ldots, l.$
  - Normalize sample weights: $w_0^{(t)} = \sum_{j=1}^{k} w_j^{(t)}, w_i^{(t)} = \frac{w_i^{(t)}}{w_0^{(t)}}, i = 1, \ldots, l.$
- Return $\sum_t^T \alpha_t b_t$

Рисунок 1.7 Псевдокод XGBoost

Узагальнюючи, XGBoost є потужним та ефективним алгоритмом машинного навчання, який дозволяє вирішувати різноманітні задачі з високою точністю. Однак, перед використанням алгоритму, необхідно ретельно налаштувати його параметри та врахувати його технічні особливості.

Random Forest - це алгоритм машинного навчання для задач класифікації, регресії та інших типів задач, що використовує ансамбль дерев рішень, розроблений Лео Брейманом (Leo Breiman) та Адель Катлер (Adele Cutler) в 2001 році [4] .

Основна ідея Random Forest полягає у створенні випадкових підвибірок з даних та випадкових підвибірок з характеристик (зазвичай, не всі доступні характеристики використовуються для кожного дерева). З цих підвибірок будується декілька дерев рішень. При класифікації або регресії нового прикладу,

кожен дерево повертає свій власний результат, а результати цих дерев об'єднуються, щоб отримати кінцевий результат. Цей процес називається багатокласовою класифікацією або ансамблюванням.

Основні параметри Random Forest включають кількість дерев, глибину дерев, кількість випадкових підвибірок характеристик та кількість об'єктів у підвибірці. Визначення правильних значень цих параметрів може допомогти у покращенні точності моделі та зменшенні перенавчання.

Алгоритм побудови дерева в Random Forest є схожим з алгоритмом побудови дерева рішень. З основною відмінністю в тому, що для кожного дерева обмежуються характеристики, які можуть бути використані для розбиття на кожному рівні дерева. Крім того, для кожного вузла в дереві, випадково обирається замість найкращого розбиття за критерієм Джині або ентропії. Це дозволяє зменшити кореляцію між деревами, що входять до ансамблю, та зменшити загальну помилку моделі.

Ще однією важливою особливістю Random Forest є можливі сть використання моделі для оцінки важливості характеристик. Важливість характеристик вимірюється тим, наскільки часто характеристика використовується для розбиття в деревах, що входять до ансамблю. Це дозволяє здійснювати відбір характеристик та виключення менш важливих характеристик, що може поліпшити точність моделі та зменшити час навчання.

Алгоритм побудови Random Forest можна описати наступним чином:

1. Задати кількість дерев, яку ми хочемо створити в ансамблі Random Forest.

2. Для кожного дерева: a. Випадково вибрати підвибірку даних з повтореннями з вихідного набору даних. b. Побудувати дерево рішень на основі цієї підвибірки.

- На кожному вузлі дерева випадковим чином вибрати певне число характеристик, за якими буде проводитися розбиття.

- Здійснити розбиття вузла на основі вибраної характеристики, яка найкраще розділяє дані.

- Продовжувати рекурсивно розбивати вузли до досягнення певної умови зупинки, наприклад, коли досягнуто максимальної глибини дерева або досягнуто мінімального числа об'єктів у вузлі.

3. Повернути ансамбль дерев у якості моделі Random Forest.

Нижче предоставлено псевдокод зі статті [3]:

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
end function
```

Рисунок 1.8 Псевдокод методу Random Forest

1.7 Показники ефективності моделей класифікації

Показники ефективності моделей класифікації визначають, наскільки точно модель може передбачити клас об'єкта. Існує багато різних показників, і кожен з них дає певну інформацію про ефективність моделі.

Матриця помилок (Confusion Matrix) - це таблиця, яка демонструє кількість правильних та неправильних прогнозів, зроблених моделлю класифікації на основі тестових даних. Вона дозволяє зрозуміти, наскільки добре модель працює на різних класах.

Матриця помилок містить чотири елементи:

- True Positive (TP) - кількість правильно визначених позитивних випадків;
- False Positive (FP) - кількість неправильно визначених позитивних випадків;
- True Negative (TN) - кількість правильно визначених негативних випадків;

• False Negative (FN) - кількість неправильно визначених негативних випадків.

Нижче надано двокласову матрицю у вигляді таблиці:

Таблиця 1.1 Вигляд двокласової confusion matrix у вигляді таблиці

		Predicted	
		Negative **(N)** **-**	Positive **(P)** **+**
Actual	Negative **-**	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive **+**	**False Negative (FN) Type II Error**	True Positive **(TP)**

Та вигляд багатоклассової матриці:

Таблиця 1.2 Вигляд багатокласової confusion matrix у вигляді таблиці

		PREDICTED classification			
	Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

За допомогою матриці помилок можна розрахувати різні метрики ефективності моделі, такі як точність, чутливість, специфічність та F1-оцінка.

Нижче надані розвідкова інформація щодо цих метрик::

1. Accuracy - відношення кількості правильно класифікованих об'єктів до загальної кількості об'єктів у вибірці.

Формула точності:

$$accuracy = (TP + TN) / (TP + TN + FP + FN) \qquad (1.1)$$

Недоліком accuracy є те, що вона може бути недостатньо інформативною у випадках, коли класи незбалансовані, тобто коли один клас має значно меншу кількість екземплярів, ніж інший. У таких випадках accuracy може бути високою, навіть якщо модель погано працює на меншому класі. Наприклад, якщо ми маємо датасет з 1000 зразків, з яких 900 належать до класу A, а 100 - до класу B, то класифікатор може досить легко досягти accuracy 90% за рахунок того, що він буде правильно класифікувати зразки класу A, але може погано працювати на зразках класу B, які він буде помилково віднести до класу A. Таким чином, метрика accuracy не відображає реальної ефективності моделі у випадку з незбалансованими даними.

2. Precision - це метрика, яка використовується для оцінки того, наскільки часто модель правильно класифікує екземпляри позитивного класу. Ця метрика розраховується як відношення кількості правильно класифікованих позитивних екземплярів до загальної кількості позитивних екземплярів, що були класифіковані моделлю:

$$Precision = TP / (TP + FP) \qquad (1.2)$$

Precision часто використовується в задачах, де важливо не допустити помилкових позитивних прогнозів. Наприклад, в медицині, якщо модель передбачає наявність захворювання, але насправді воно відсутнє, то це може призвести до неправильного лікування. У такому випадку ми більше зацікавлені у високій precision, ніж у високій чутливості (recall).

Однак, precision може бути обманливою метрикою, особливо в задачах з незбалансованими даними, коли кількість екземплярів позитивного класу дуже мала порівняно з екземплярами негативного класу. У таких випадках модель може бути дуже точною в класифікації негативних екземплярів, але не здатна коректно класифікувати позитивні. Тому, перед використанням precision як метрики, необхідно детально проаналізувати дані та визначити, чи є вони збалансованими, і чи необхідно використовувати інші метрики для оцінки ефективності моделі.

3. Повнота (Recall), також відома як чутливість (sensitivity) або True Positive Rate (TPR), є метрикою, що вимірює здатність класифікатора правильно визначати позитивні зразки з усіх дійсно позитивних зразків. Вона вказує на те, яка частка позитивних зразків була виявлена класифікатором.

Повнота обчислюється за формулою:

$$Recall = TP / (TP + FN) \quad (1.3)$$

Це відношення показує, яка частка позитивних зразків була виявлена класифікатором. Чим більше значення повноти, тим краще класифікатор виявляє позитивні зразки. Високе значення повноти означає, що класифікатор мало пропускає позитивні зразки, але водночас може призводити до більшої кількості неправильно класифікованих негативних зразків.

Повнота є важливою метрикою в задачах, де виявлення позитивних зразків має високий пріоритет, наприклад, у виявленні хвороб, де недіагноз може мати серйозні наслідки.

4. F1-score (F1-міра) є гармонічним середнім між точністю та повнотою, що використовується для вимірювання точності бінарної класифікації. Це є зважено середнє між точністю і повнотою, і він приймає значення між 0 та 1.

Формула F1-score:

$$F1 - score = 2 * (precision * recall) / (precision + recall) \quad (1.4)$$

F1-міра використовується в тих випадках, коли точність та повнота мають різне значення. Якщо точність висока, а повнота низька, то значення F1-міри буде низьким, а якщо точність низька, а повнота висока, то F1-міра також буде низькою. F1-міра зазвичай використовується, коли дані є незбалансованими, тобто один клас має значно більше прикладів, ніж інший. Наприклад, якщо ми маємо 1000 зображень, і 950 з них належать до класу "кіт", а 50 до класу "собака", то модель може класифікувати всі зображення як "кіт", отримуючи точність 95%. Однак, це буде неправильною моделлю, тому що вона не може відрізнити між котами та собаками. У цьому випадку важливішою буде повнота, оскільки ми хочемо, щоб модель змогла визначати обидва класи. F1-міра враховує як точність, так і повноту, тому вона є кращою метрикою в таких випадках.

Недоліком F1-міри є те, що вона не враховує зразки, які були класифіковані правильно, але помилково визначені як інший клас. В такому випадку метрика може бути низькою, навіть якщо модель може коректно визначати більшість

зразків. Також F1-міра не дозволяє враховувати більшість поганих результатів в одному з класів, що може бути важливою інформацією у деяких задачах. Наприклад, якщо ми працюємо з даними про пацієнтів з деяким захворюванням, то помилкове визначення хворих пацієнтів як здорових може бути критично важливим.

Перевагою F1-міри є те, що вона є зваженою мірою, яка враховує обидві метрики - точність та повноту. Це дозволяє отримати більш точну інформацію про ефективність моделі в задачах класифікації з незбалансованими даними. Крім того, F1-міра є доброю метрикою, коли точність та повнота мають приблизно однакове значення.

Загалом, F1-міра є корисною метрикою, яка дозволяє враховувати як точність, так і повноту, та допомагає визначити ефективність моделі в задачах класифікації з незбалансованими даними.

6. AUC-ROC (Area Under the Receiver Operating Characteristic Curve) - це метрика, яка використовується для оцінки якості бінарної моделі класифікації. Ця метрика вимірює площу під кривою ROC (Receiver Operating Characteristic), яка є графіком, що показує залежність між True Positive Rate (TPR) і False Positive Rate (FPR) при зміні порога відсічення.

TPR – це вже відома нам метрика, Повнота (Recall) (1.3), яку ми вже розглянули. FPR - це відношення неправильно класифікованих негативних прикладів до загальної кількості негативних прикладів в тестовому наборі.

Формули для AUC-ROC:

$$TRP = \frac{TP}{TP+FN} \tag{1.5}$$

$$FRP = FP/(FP + TN) \tag{1.6}$$

Крива ROC дає можливість оцінити те, як добре модель розділяє позитивні та негативні класи, в залежності від значення порога відсічення. Ідеальна модель буде мати площу під кривою дорівнює 1, тоді як модель, яка не відрізняється від випадкового вибору, матиме площу дорівнює 0.5.

$$AUC - ROC = \int_0^1 TPR\big(f(x)\big)dFRP(fx)) \tag{1.7}$$

AUC-ROC дуже корисна метрика в разі, коли класи в даних незбалансовані, тобто один з класів має більше прикладів, ніж інший. Оскільки ця метрика не залежить від пропорцій класів, вона може допомогти уникнути помилкового

враження про те, що модель є досить точною, тоді як насправді вона дає низьку точність на менш представлених класах.

До переваг AUC-ROC можна віднести її стійкість до шуму та зміни порога відсічення, що дозволяє використовувати цю метрику в різних ситуаціях. Також важливою перевагою є те, що AUC-ROC можна використовувати для порівняннярізних моделей, які можуть мати різні значення порога відсічення.

Недоліки AUC-ROC полягають у тому, що вона не дає змоги оцінити ефективність різних класифікаторів на основі різних порогів відсічення на прикладах з різною важливістю, оскільки площа під кривою є вагомою метрикою, і значення цієї метрики може бути неадекватним для визначення ефективності класифікатора на деяких конкретних важливих прикладах. Також AUC-ROC може не підходити для задач, де дуже важливо, щоб модель правильно класифікувала певний клас, який є меншим за інші класи в тестовому наборі.

7. Log loss - це метрика, яка використовується для оцінки точності класифікаторів. Вона є функцією відстані між прогнозованими й фактичними значеннями. Ця метрика вимірюється в діапазоні від 0 до нескінченності, де 0 позначає ідеальну точність, а значення, які прямують до нескінченності, вказують на повну невідповідність прогнозів реальності.

Формула для розрахунку log loss (1.6):

$$log\ loss = -(1/N) * \sum[y * log(p) + (1 - y) * log(1 - p)] \quad (1.6)$$

де:

- N - загальна кількість прикладів у наборі даних
- y - фактична мітка класу (0 або 1)
- p - передбачена ймовірність належності до класу 1 за моделлю

Сума береться по всіх прикладах в наборі даних, і для кожного прикладу обчислюється вираз y * log(p) + (1 - y) * log(1 - p). Потім отримані значення сумуються і помножуються на -(1/N), що дає загальну логарифмічну втрату для набору даних.

Log loss доречно використовувати в задачах бінарної та багатокласової класифікації. Його головною перевагою є те, що він дуже чутливий до неправильних передбачень, тобто якщо класифікатор вважає один клас більш ймовірним, але насправді він належить до іншого класу, то це значно підвищить

значення log loss. Завдяки цьому метрика дає важливу інформацію про те, як далеко класифікатор відхиляється від правильних відповідей. Однак він також може бути чутливим до перевірочних даних та до великої кількості класів.

Кожен з цих показників має свої переваги і недоліки, тому важливо використовувати декілька показників для оцінки ефективності моделі і приймати рішенняв залежності від контексту задачі. Наприклад, якщо ми важливіше уникнути помилок визначення позитивних об'єктів, то точність і точність класифікації позитивних об'єктів (precision) можуть бути більш інформативними показниками. У той же час, якщо ми важливіше забезпечити, щоб не було пропущених позитивних об'єктів, тоді повнота (recall) може бути більш інформативною метрикою.

Висновок

В ході аналізу та вивчення теоретичного матеріалу дипломної роботи були розглянуті наступні теми:

Аналіз та дослідження банківських транзакції та особливості їх набору даних, що впливають на методи та якість класифікації.

Методи підвищення якості прогнозу моделей класифікації, їх вплив на моделі, дані та результат, їх недоліки, переваги та чим кожен з них відрізняються від інших.

Також розглянуті були нерівномірність розподілу класів та проблема нерівномірної класифікації у задачах класифікації, іх вплив на моделі прогнозування.

Вивчені та проаналізовані методи балансування даних у незбалансованому наборі даних, такі як oversampling, undersampling, SMOTE та ADASYN, їхпереваги, недоліки та в яких задачах їх краще використовувати.

Методи групування даних, такі як алгоритм К-середніх та нечіткої кластерізації, іх зміст, призначення до роботи та чим кожен з них відрізняються від іншого

Методи класифікації незбалансованих даних, такі як XGBoost та Random Forest.

Показники ефективності або метрики моделей класифікації такі як Accuracy, Precision, AUC-ROC, Log-loss та F1-score, проаналізовано та вивчено чим вони відрізняються та в яких задачах їх більш корректно застосовувати.

Після вивчення теоретичного матеріалу ми ознайомилися з проблемами незбалансованих даних у задачах класифікацій. Отриманий теоретичний базис надає можливість розробити ефективний підхід до вирішення задачі прогнозування шахрайства на основі незбалансованих даних. У наступних розділах роботи буде розглянуто практичне застосування цих методів та технік для побудови класифікаційної моделі, яка зможе ефективно прогнозувати випадки шахрайських транзакцій враховуючи незбалансованість даних.

СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Постановка задачі

2.1.1 Мета дослідження

Метою даного дослідження полягає в застосуванні та порівнянні методів класифікації незбалансованих наборів даних для використання в задачах виявлення фроду. Конкретніше, метою є розробка ефективного підходу до прогнозування транзакцій на наявність фроду на основі незбалансованих даних та порівняння результатів з використанням різних методів, таких як Random Forest, XGBoost. Дослідження має на меті зрозуміти, які методи класифікації можуть бути ефективнішими при роботі з незбалансованими даними в контексті задач виявлення фроду.

2.1.2 Задачі дослідження

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Розглянути теоретичні аспекти класифікації даних, зокрема, методи роботи з незбалансованими даними.

2. Оглянути різноманітні методи класифікації незбалансованих наборів даних, включаючи методи, які засновані на деревах рішень, нейронних мережах, ансамблеві методи та інші.

3. Дослідити існуючі наукові роботи, присвячені використанню методів класифікації для задач виявлення фрода.

4. Описати методи XGBoost та Random Forest та порівняти їх.

5. Розробити експериментальну частину, в рамках якої буде проведено експеримент з використанням методу XGBoost та Random Forest на даних з задачі виявлення фрода.

6. Проаналізувати результати експерименту та зробити висновки про ефективність методів XGBoost та Random Forest у класифікації незбалансованих наборів даних в задачах виявлення фрода.

7. Зробити висновки щодо зробленої роботи та актуальності та практичних застосувань цих методів у задачах на виявляння фроду.

2.2 Середа програмування для дослідження роботи

Для прогнозування фроду (шахрайства) існує багато середовищ програмування, які можна використовувати. Вибір конкретного середовища залежить від ваших вимог, знань та вподобань. Ось декілька популярних середовищ програмування, які часто використовуються для прогнозування фроду:

1. Python: Python є однією з найпоширеніших мов програмування для аналізу даних та машинного навчання. Він має багато потужних бібліотек, таких як scikit-learn, TensorFlow, PyTorch і XGBoost, які допомагають у побудові моделей прогнозування фроду.

2. R - мова програмування та середовище для статистичного аналізу та візуалізації даних. Вона також має багато пакетів для машинного навчання, таких як caret, randomForest і xgboost, які можуть бути використані для прогнозування фроду.

3. MATLAB - інтерактивна система для чисельних обчислень та аналізу даних. Вона має багато інструментів для машинного навчання, таких як Statistics and Machine Learning Toolbox, які можуть бути використані для прогнозування фроду.

4. SAS - програмний пакет для статистичного аналізу та аналізу даних. Він має спеціалізовані інструменти для прогнозування шахрайства, такі як SAS Fraud Framework, який надає засоби для виявлення та прогнозування шахрайства.

5. Spark - це фреймворк для обробки великих обсягів даних та аналізу в розподіленому середовищі. Він має вбудовану бібліотеку машинного навчання (MLlib), яка надає інструменти для побудови моделей прогнозування фроду на розподілених обчислювальних кластерах.

Крім того, існують інші середовища програмування, які також можуть бути використані для прогнозування фроду, такі як Julia, Java, C++, і так далі.

Ця дипломна робота була виконана виключно у середовищі Python.

Основні бібліотеки, які були використані:

- Pandas: ця бібліотека дозволяє працювати з структурованими даними, такими як табличні дані, і забезпечує різноманітні функції для маніпуляції, фільтрації, об'єднання та агрегації даних. Pandas надає багато функціональностей для ефективної роботи з даними, таких як читання і запис даних з різних форматів (наприклад, CSV, Excel, SQL), обробка пропущених значень, обчислення статистичних показників, візуалізація даних та багато іншого.

- NumPy (Numerical Python) - бібліотека, яка надає підтримку для обчислення матриць, векторів та інших багатовимірних масивів. Вона є однією з основних бібліотек для наукових обчислень у Python і надає широкий набір функцій і операторів для роботи з числовими даними. NumPy надає ефективні структури даних для зберігання і маніпуляції числової інформації, включаючи масиви, вектори і матриці. Вона також містить функції для виконання математичних операцій, лінійної алгебри, статистики, обробки зображень та багато іншого.

- Scikit-learn (також відома як sklearn) – ця бібліотека надає широкий набір інструментів і функцій для завдань класифікації, регресії, кластеризації, зменшення розмірності, підбору параметрів та інших завдань, пов'язаних з машинним навчанням.

Scikit-learn побудована на основі інших популярних бібліотек, таких як NumPy та SciPy, і надає простий у використанні та консистентний інтерфейс для роботи з моделями машинного навчання. Вона містить реалізації різних алгоритмів, включаючи методи на основі дерев рішень, метод опорних векторів, навчання з учителем та навчання без учителя, а також інструменти для попередньої обробки даних, оцінки моделей, підбору гіперпараметрів і валідації.

- Matplotlib та Seaborn: Matplotlib є основною бібліотекою для створення графіків і візуалізації даних у Python. Вона надає широкий набір функцій для створення різних типів графіків, включаючи лінійні графіки, стовпчасті діаграми, кругові діаграми, розсіювальні графіки, графіки контуру та інші. Matplotlib дозволяє налаштовувати різні аспекти графіків, включаючи заголовки, мітки осей,

легенду, колірну палітру та інші елементи, що допомагають передати інформацію візуально.

Seaborn - це вищорівнева бібліотека для створення стильних та привабливих графіків у Python. Вона побудована на основі Matplotlib і надає більш простий та зручний інтерфейс для створення графіків зі стандартними налаштуваннями. Seaborn надає спеціалізовані функції для створення графіків, які зазвичай використовуються для візуалізації статистичних даних, таких як графіки розподілу, ящикові діаграми, графіки кореляції та інші. Вона також пропонує красиві стилі оформлення графіків, що допомагають створювати привабливі та професійні візуалізації.

Обидві бібліотеки, Matplotlib і Seaborn, дозволяють створювати графіки з великою гнучкістю і настроюваннями, що дозволяє відтворювати дані у вигляді зрозумілого та інформативного візуального представлення.

Використовуючи мову Python та вищерозглянуті бібліотеки, почнемо виконувати EDA.

2.3 Exploratory Data Analysis

Основним набором даних для дослідження та прогнозування буде датасет «transactions_train.csv», отриманий з сайту Kaggle. У ньому знаходяться дані щодо банківських транзакцій та їх легітимності.

Нижче будуть надані ознаки цього датасету:

Таблиця 2.1 Перелік ознак датасету

Step	відображає одиницю часу в реальному світі. У цьому випадку 1 крок - це 1 година часу
Type	CASH-IN, CASH-OUT, DEBIT, PAYMENT та TRANSFER.
Type	сума операції в національній валюті.
nameOrig	клієнт, який почав операцію.
oldbalanceOrig	початковий баланс до операції.
newbalanceOrig	баланс клієнта після транзакції.
nameDest	ідентифікатор одержувача транзакції.
oldbalanceDest	початковий баланс одержувача до транзакції.
newbalance	баланс одержувача після транзакції.

Dest	
isFraud	чи є шахрайством або ні.

Ознайомившись з ознаками даних цього датасету, потрібно завантажити його. Подальші завантаження та відображення ознак та даних датасету будуть виконані завдяки бібліотеці «Pandas».

Виведемо перші три строки датасету для перевірки, чи правильний датасет було завантажено та чи усі ознаки присутні у ньому:

	step	type	amount	nameOrig	oldbalanceOrig	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1

Рисунок 2.1 Дані датасету «transactions_train.csv»

Як бачимо, усі ознаки та дані були завантажені корректно. Далі нам потрібно вивести типи ознак, щоб розуміти які ознаки є чисельними та категоріальними:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6351193 entries, 0 to 6351192
Data columns (total 10 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrig  float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
dtypes: float64(5), int64(2), object(3)
memory usage: 484.6+ MB
```

Рисунок 2.2 Типи ознак

Бачимо, що більшість ознак є числовими, але 3 з них – type, nameOrig та nameDest є категоріальними, тому їх потім потрібно буде за допомогою LabelEncoder закодувати у числові.

Далі розрахуємо кількість пропущених значень (NaN) в кожному стовпці (колонці) датафрейму і повертає ці значення у вигляді масиву (array). Для кожного стовпця масив містить кількість пропущених значень. Наприклад, якщо значення для першої колонки (стовпця) рівне 100, то це означає, що в цьому стовпці є 100 пропущених значень.

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Рисунок 2.3 Перевірка пропущених значень

Як бачимо, пропущенних значень у цьому датасеті ми не маємо.

Далі перевіримо кількість дубльованих записів у наборі даних train.

```
Out[73]: 0
```

Рисунок 2.4 Кількість дубльованих значень

Дублікатів теж немає, що спрощує аналіз даних.

Тепер нам потрібно вичислити skewness(коефіцієнт асиметрії). В машинному навчанні skewness використовується для оцінки форми розподілу даних та визначення його симетрії або асиметрії. Якщо розподіл даних має високий рівень skewness, це може вказувати на те, що дані не є рівномірно розподіленими та мають виразну симетрію, що може впливати на точність моделі машинного навчання. Отже, для покращення точності моделі машинного навчання може бути корисним провести аналіз skewness і зробити корекцію даних, щоб знизити або усунути асиметрію розподілу даних. Таблиця коефіцієнта асиметрії містить перші 10 стовпців з найвищими значеннями коефіцієнта асиметрії у спадаючому порядку. Коефіцієнт асиметрії вказує на розподіл даних в стовпці: якщо коефіцієнт асиметрії дорівнює нулю, то розподіл є симетричним; якщо він менше нуля, то розподіл має ліву асиметрію, а якщо більше нуля, то праву. Значення коефіцієнта асиметрії від 0,5 і вище вказують на значну асиметрію даних.

Таблиця 2.2 Коефіцієнт асиметрії

	skew
amount	31.050928
isFraud	28.635901
oldbalanceDest	19.934164
newbalanceDest	19.362310
oldbalanceOrig	5.243790
newbalanceOrig	5.172421
step	0.338249

Як бачимо, коефіцієнт ассиметрії дуже великий, його потрібно відкоригувати перед тим, як робити модель прогнозування.

Для цього спочатку, для кращого розуміння різниці між невідкоригованими даними та відкоригованими, відобразимо графіки нинішніх даних:

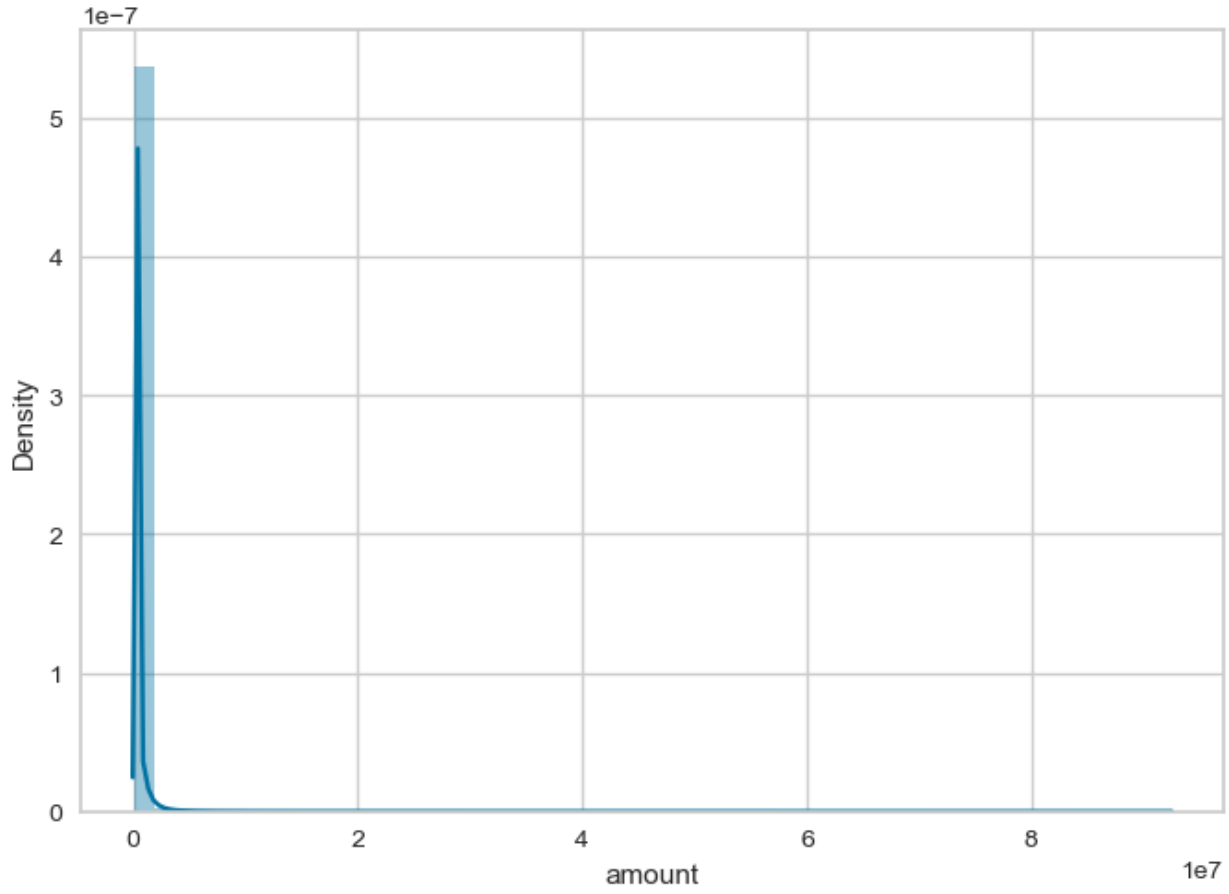

Рисунок 2.5 Відображення асиметрії даних

Виконаємо корекцію розподілу значень стовпця amount за допомогою логарифмування:

```
КА до логарифмування: 31.050928455018084
КА після логарифмування: -0.5549658313745637
```

Рисунок 2.6 Відкореговані дані

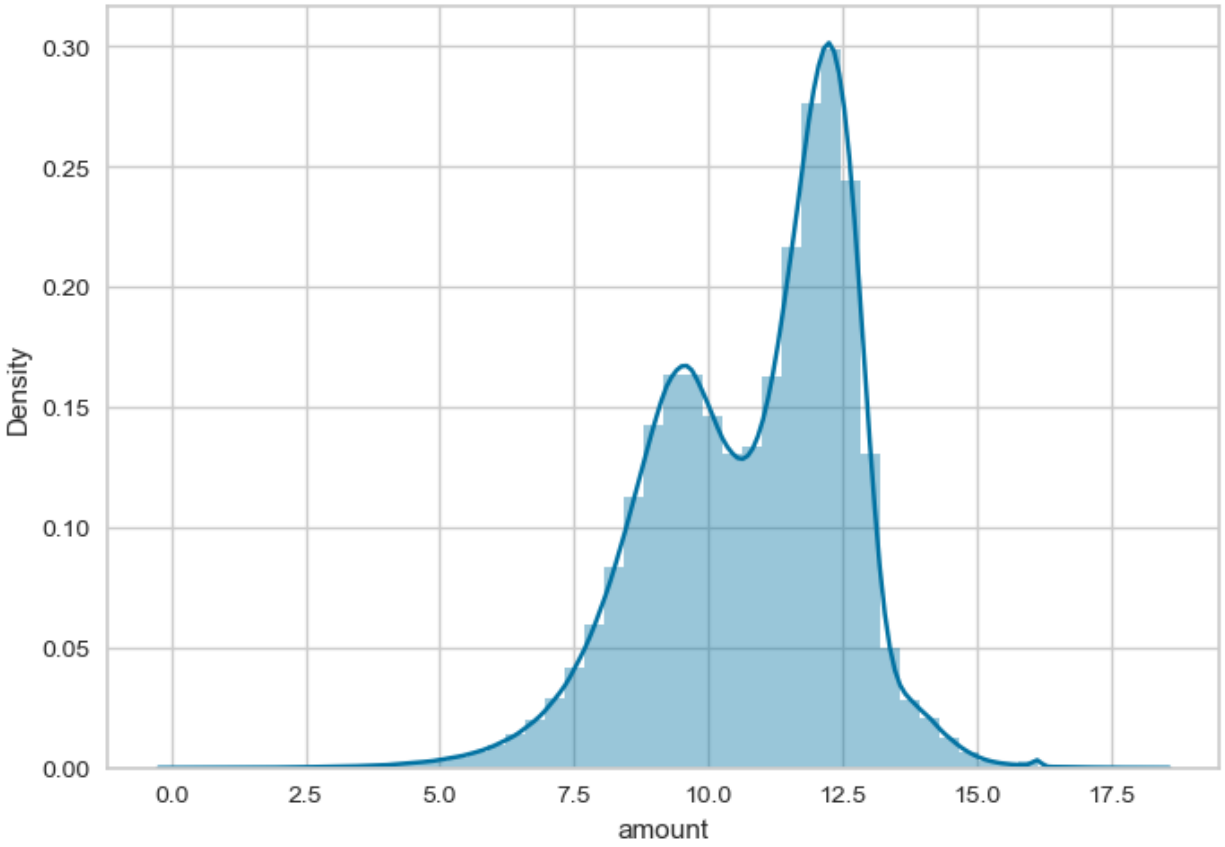

Рисунок 2.7 Графік після логарифмування

Отже, після логарифмування, значення коефіцієнта асиметрії зменшилось, що означає, що розподіл став менш скошеним та більш симетричним.

Далі обчислимо коефіцієнт ексцесу. Обчислення куртозису для кожного стовпця дозволяє оцінити, наскільки вибірка відрізняється від нормального розподілу. Значення куртозису можуть бути позитивними або від'ємними.

Таблиця 2.3 Коефіцієнти ексцесу

	Kurtosis
step	0.246047
amount	1803.410673
oldbalanceOrig	32.875430
newbalanceOrig	32.003795
oldbalanceDest	950.015902
newbalanceDest	863.076045
isFraud	818.015079

Значення куртозису більше 0 вказує на більш "концентрований" розподіл з тяжкими хвостами, тоді як значення менше 0 вказує на менш "концентрований" розподіл з меншими хвостами. Значення навколо 0 вказують на близькість до нормального розподілу.

1. Для стовпця "step" куртозис дорівнює 0.246047, що вказує на наближеність до нормального розподілу.
2. Для стовпця "amount" куртозис дорівнює 1803.410673, що вказує на дуже велику хвостатість розподілу. Це означає, що значення "amount" розподілені широко з великими хвостами, що може вказувати на наявність викидів або аномалій у даних.
3. Для стовпців "oldbalanceOrig", "newbalanceOrig", "oldbalanceDest" і "newbalanceDest" куртозиси дорівнюють відносно невеликим значенням, що вказує на наближеність до нормального розподілу. Це означає, що значення в цих стовпцях мають меншу хвостатість і більшу концентрацію навколо середнього значення.
4. Для стовпця "isFraud" куртозис дорівнює 818.015079, що вказує на велику хвостатість розподілу.

Тепер обчислемо дисперсії для всіх стовпчиків. Аналіз дисперсії допомагає зрозуміти варіативність та розподіл даних у відповідних стовпцях вашого датафрейму. Ця інформація може бути використана для подальшого аналізу, видалення аномалій, вибору ознак для моделювання та виконання інших завдань аналізу даних.

Таблиця 2.4 Дисперсія стовбців

	var
isFraud	1.213571e-03
step	1.990008e+04
amount	3.643704e+11
oldbalanceOrig	8.351864e+12
newbalanceOrig	8.561904e+12
oldbalanceDest	1.155268e+13
newbalanceDest	1.350043e+13

Зі значень дисперсії можна зробити наступні спостереження:

1. isFraud: Це стовпець, що вказує на наявність або відсутність шахрайства. Значення дисперсії дуже низькі (1.213571e-03), що може свідчити про те, що дані в цьому стовпці майже однакові або майже не змінюються. Відсутність великої варіативності в цьому стовпці може бути пов'язана з тим, що більшість транзакцій не є шахрайськими.

2. step: Це стовпець, який вказує на часовий крок транзакції. Значення дисперсії досить великі (1.990008e+04), що свідчить про значну різноманітність значень і можливо про довгий часовий проміжок між окремими транзакціями.

3. amount, oldbalanceOrig, newbalanceOrig, oldbalanceDest, newbalanceDest: Ці стовпці пов'язані з сумами грошей або балансами на рахунках. Значення дисперсії у всіх цих стовпцях дуже великі, що свідчить про значну варіативність у сумах та балансах. Це може бути пов'язано з різними типами транзакцій та рахунків, які мають різні розміри та зміни у значеннях.

Зі спостережень дисперсій можемо зробити висновок, що аномалій не знайдено.

Далі обчислемо IQR або міжквартильний діапазон. Обчислення IQR дозволяє визначити "нормальний" діапазон значень у стовпці. Зазвичай вважається, що значення, що виходять за цей діапазон, можуть бути потенційними викидами або аномаліями.

Таблиця 2.5 IQR

step	179.00
amount	195326.90
oldbalanceOrig	107346.00
newbalanceOrig	144365.15
oldbalanceDest	943866.12
newbalanceDest	1112791.08
isFraud	0.00
dtype: float64	

На підставі отриманих значень IQR для кожного стовпця можна зробити наступні висновки:

1. step: Розмах значень стовпця step становить 179, що означає, що дані мають відносно невелику варіабельність в цьому стовпці.

2. amount: Розмах значень стовпця amount досить великий і становить 195,326.90. Це свідчить про значну варіативність суми транзакцій.

3. oldbalanceOrig та newbalanceOrig: Ці стовпці також мають значну варіабельність, оскільки їх розмахи становлять відповідно 107,346.00 і 144,365.15. Це означає, що вихідні та нові баланси походять з різних діапазонів значень.

4. oldbalanceDest та newbalanceDest: Ці стовпці також мають значну варіабельність, оскільки їх розмахи становлять відповідно 943,866.12 і 1,112,791.08. Це означає, що старі та нові баланси отримувачів також знаходяться в широкому діапазоні значень.

Ми можемо видалити викиди із датасету за допомогою міжквантильного діапазону, але це може видалити значення з шахрайських транзакцій, що може зменшити шанси на виявлення фроду під час прогнозування. Але ми можемо використовувати алгоритми, стійкі до викидів, такі як дерева рішень, випадковий ліс або алгоритм XGBoost, що допоможе нам не втрачати значень для прогнозування фроду.

Далі створимо pair plot, який використовує бібліотеку Seaborn для візуалізації взаємозв'язків між змінними з датасету з перших 50 000 рядків.

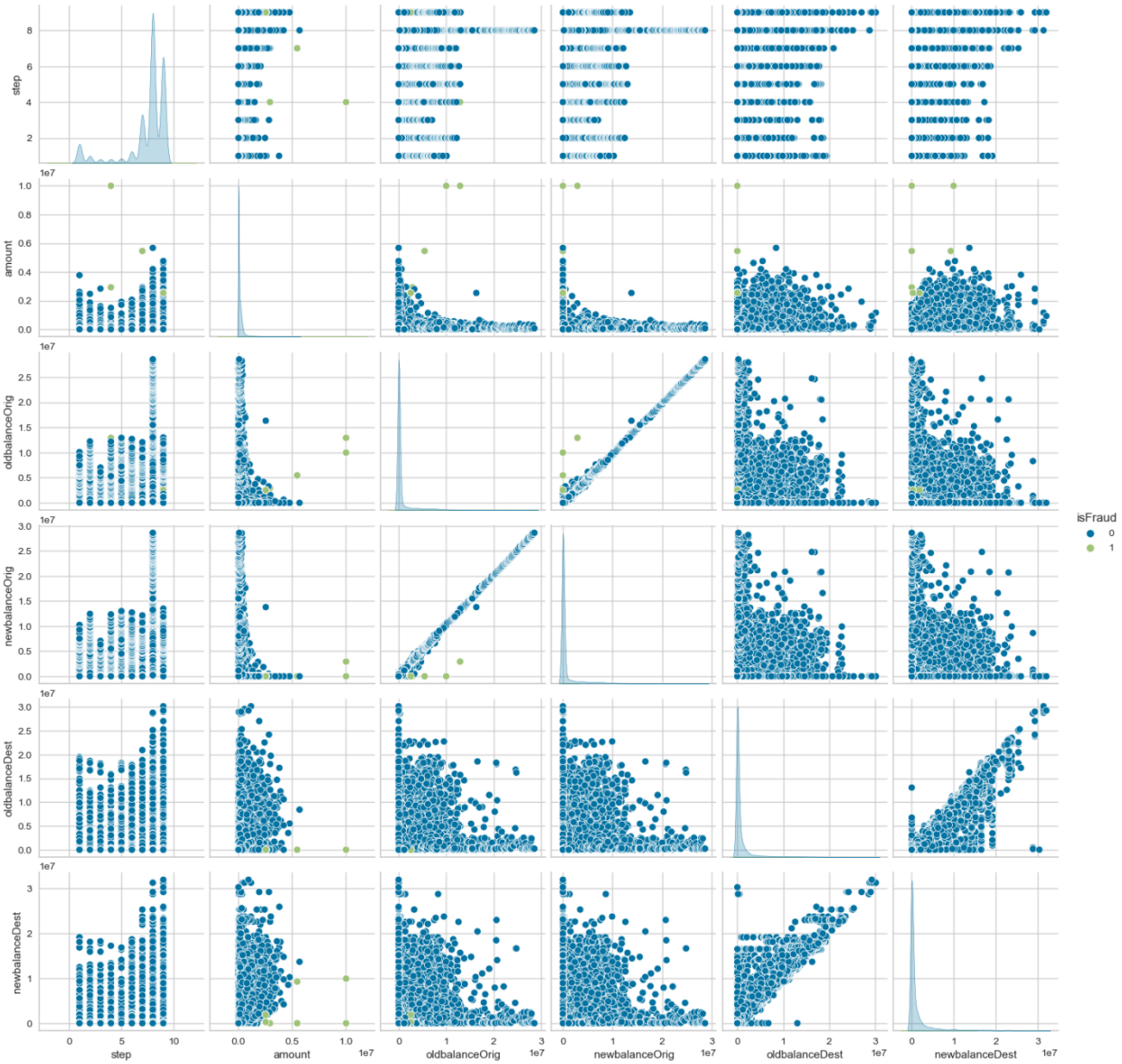

Рисунок 2.8 Графік взаємозв’язків ознак датасету до логарифмування

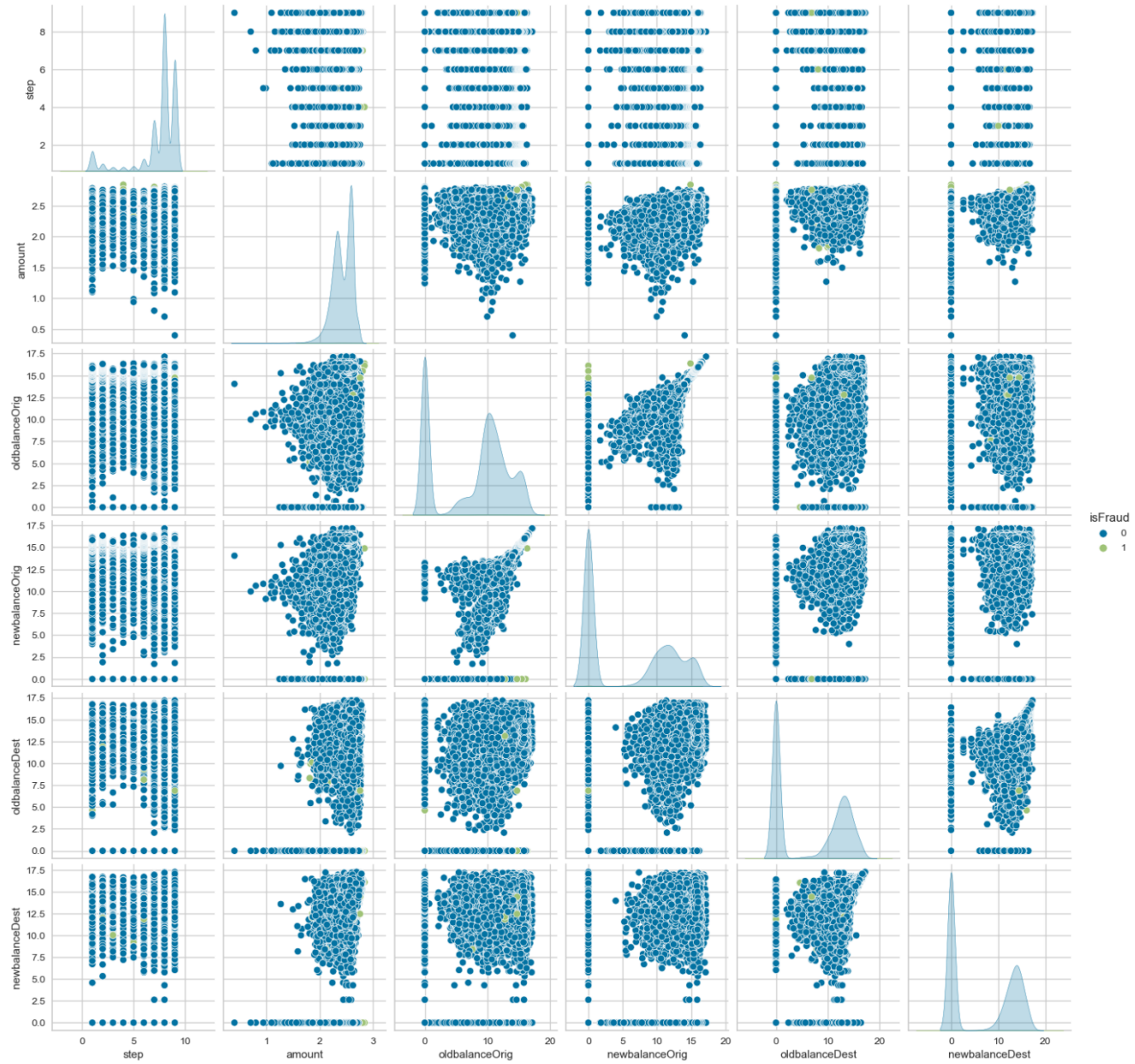

Рисунок 2.9 Графік взаємозв’язків ознак датасету після логарифмування

Як можемо побачити, newbalanceOrig та oldbalanceOrig мали сильні взаємозв’язки до логарифмування, також значимий зв’язок мали newbalanceDest та oldbalanceDest. Сильний взаємозв'язок між цими змінними може бути важливим для аналізу фінансових транзакцій, наприклад, це може свідчити про здійснення переказів коштів між різними рахунками або зміну балансу після здійснення операцій, але через логарифмування ми втратили деяку частину зв’язку, тому кращим варіантом буде логарифмувати лише ознаку amount, бо вона має найбільший коефіцієнт ассиметрії.

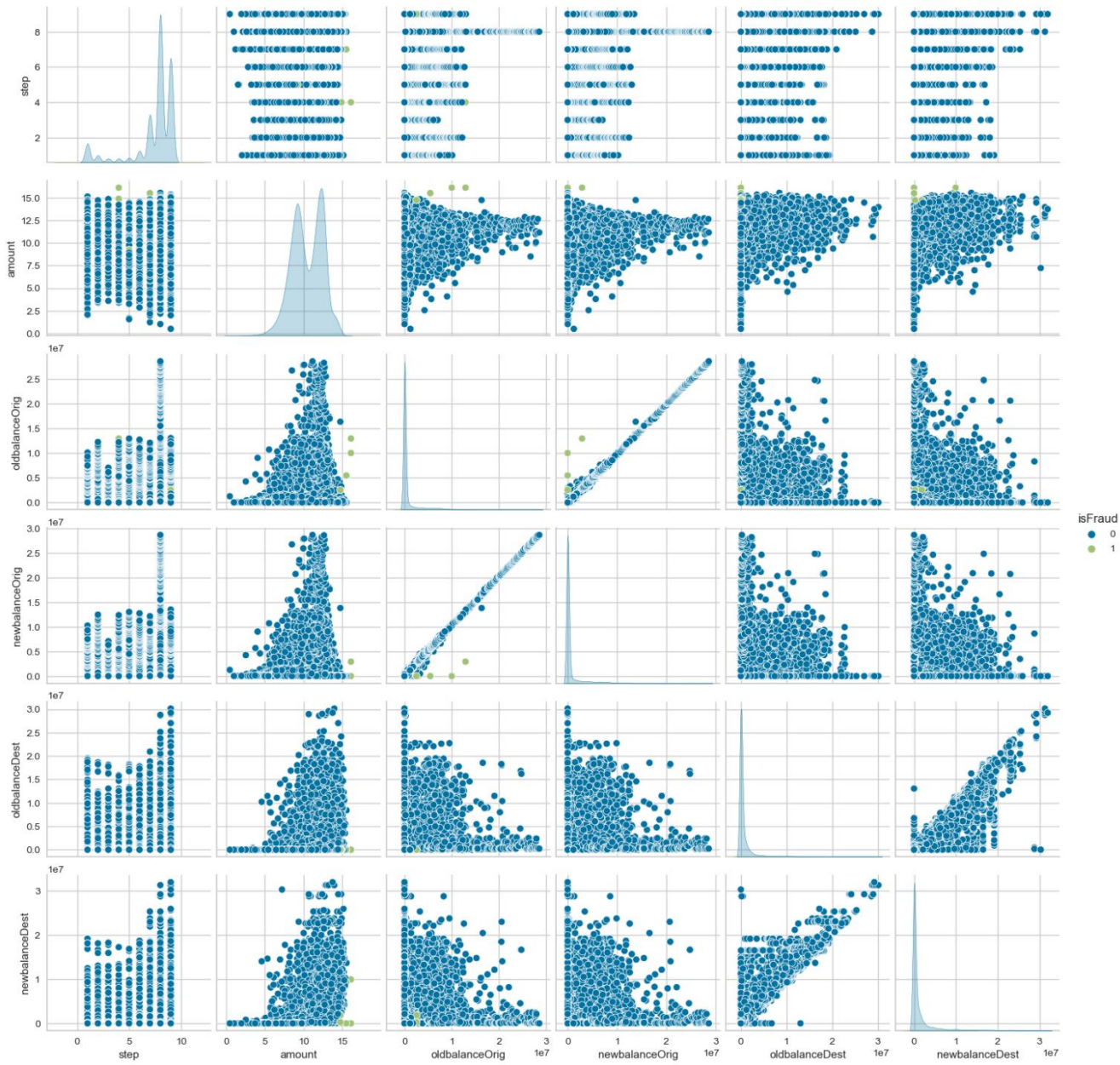

Рисунок 2.10 Графік взаємозв’язків ознак датасету після логарифмування лише ознаки amount

Далі створимо кореляційну матрицю для датасету і візуалізує її в графічному вигляді за допомогою функції corr_plot з бібліотеки klib.

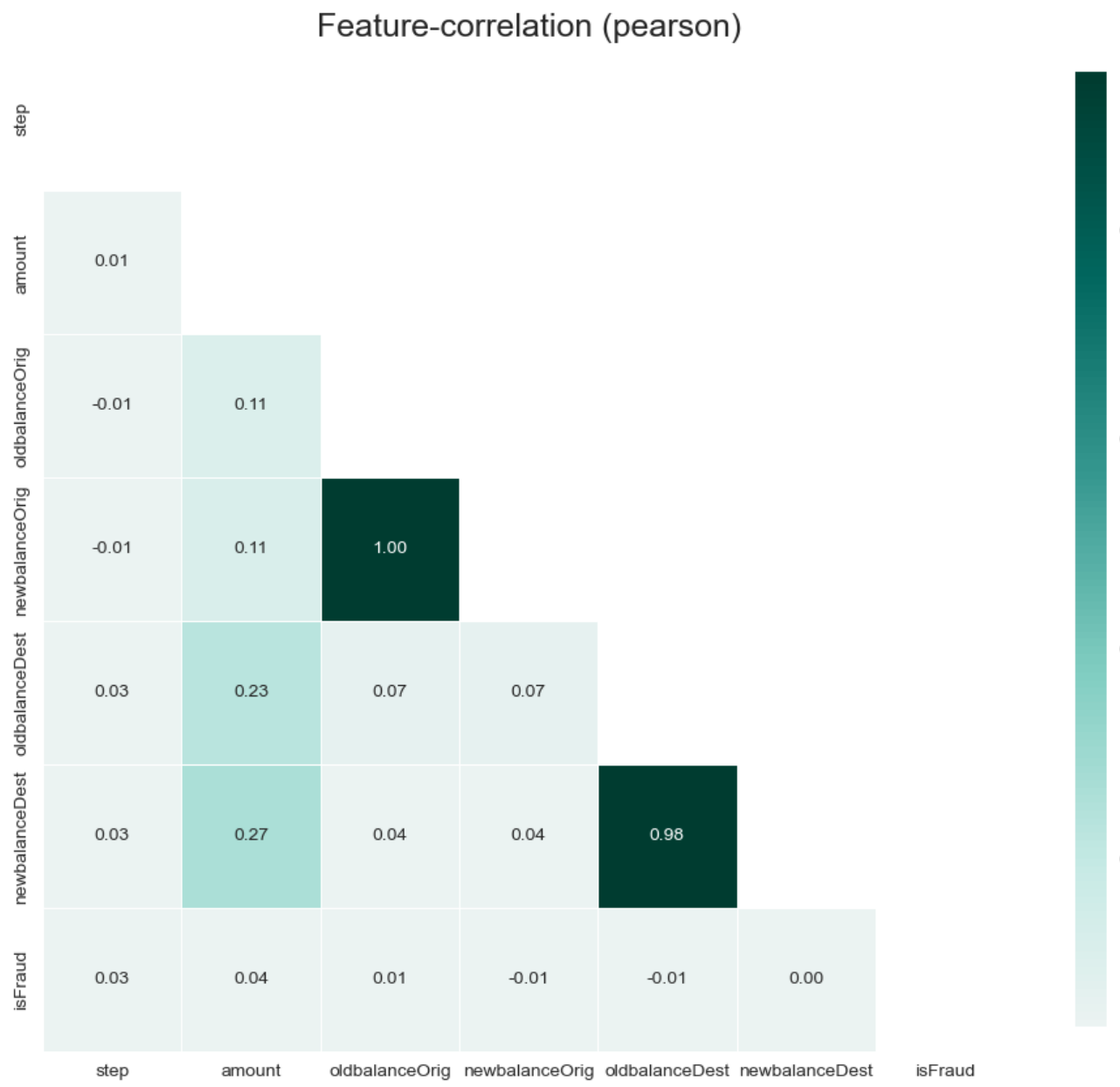

Рисунок 2.11 Графік кореляції Пірсона

Дійсно бачимо, взаємозв’язок newbalanceOrig та oldbalanceOrig є майже 100%, що означає що кожна зміна даних у однієї з цих двох ознак прямо впливає на іншу.

Далі змінемо тип даних колонок у датафсеті на числові та категорійні за допомогою методу .apply() з бібліотеки Pandas. Це може бути корисно, якщо деякі стовпці були помилково розпізнані як рядкові об'єкти або категорії, і їх потрібно перетворити на числові типи, щоб проводити числові операції і аналіз.

Далі створимо змінні x та y, що містять вхідні дані та відповідні мітки класів для задачі класифікації.

Далі створимо дві змінні: cat_columns та num_columns. Змінна cat_columns містить назви усіх категорійних колонок у X, а змінна num_columns містить назви усіх числових колонок. Ці змінні можуть бути використані для подальшої обробки та аналізу даних відповідного типу.

Далі створимо 6 гістограм, по одній для кожної змінної з числовими значеннями (що містяться в змінній num_columns) з даних навчального набору даних. Кожна гістограма містить розподіл значень змінної, показаний за допомогою кількох стовпчиків (бінів), де кожен стовпчик відповідає певному діапазону значень змінної. Також на гістограмі відображений неперервний розподіл щільності (kde), який дозволяє оцінити швидкість зміни частоти виникнення значень. У кожній гістограмі мітки на осі x показують значення змінної, а на осі y - частоту виникнення значень.

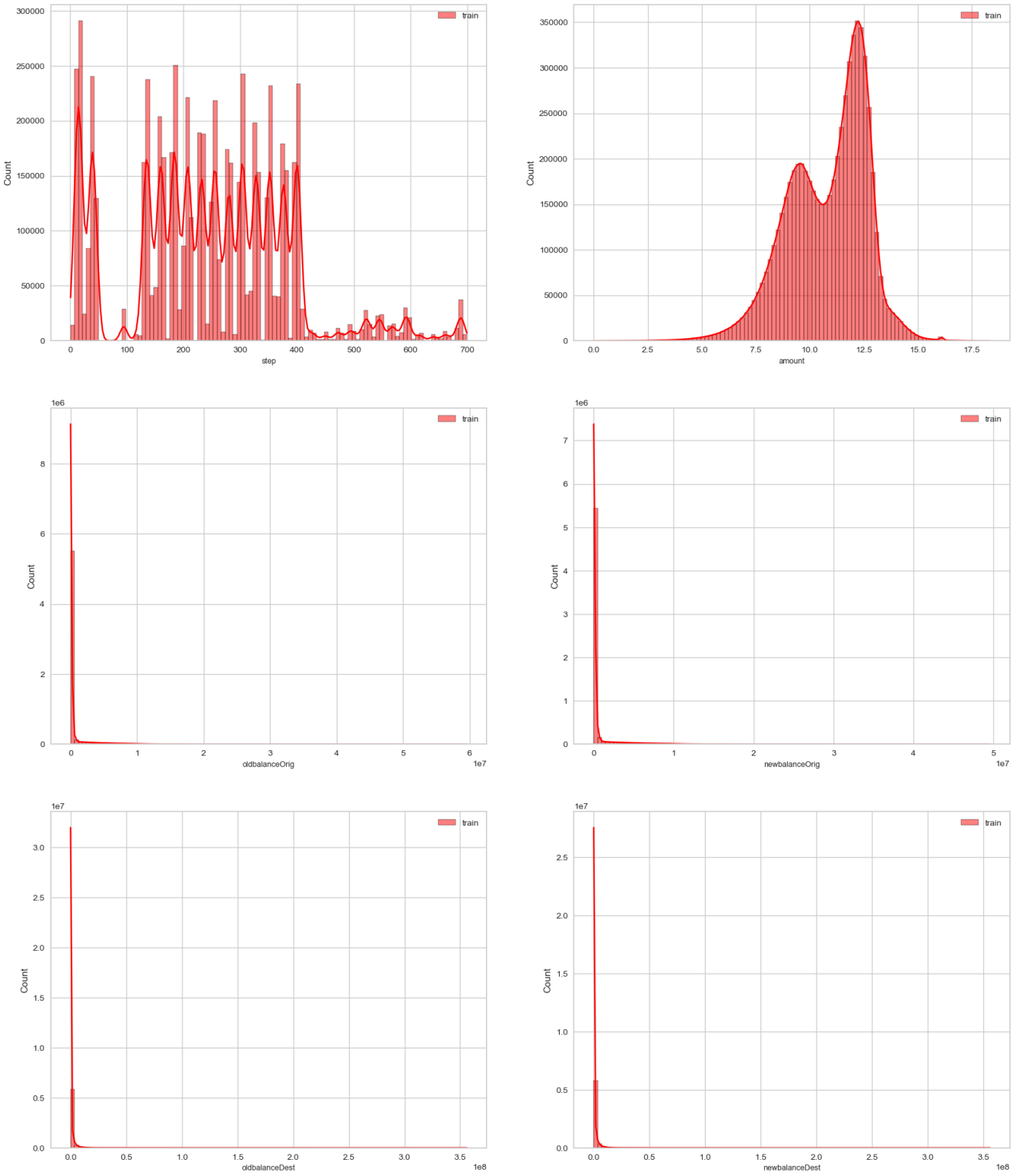

Рисунок 2.12 Гістограми розподілів змінної

Далі ще раз обчислимо кореляцію між числовими ознаками у наборі даних, застосовуючи градієнтний колір для більш наочного показу кореляційних коефіцієнтів :

	step	amount	oldbalanceOrig	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.007333	-0.009113	-0.009201	0.028303	0.026508	0.025495
amount	0.007333	1.000000	0.106908	0.111480	0.228012	0.266220	0.039389
oldbalanceOrig	-0.009113	0.106908	1.000000	0.998857	0.066301	0.042019	0.009226
newbalanceOrig	-0.009201	0.111480	0.998857	1.000000	0.067852	0.041853	-0.008322
oldbalanceDest	0.028303	0.228012	0.066301	0.067852	1.000000	0.976550	-0.005657
newbalanceDest	0.026508	0.266220	0.042019	0.041853	0.976550	1.000000	0.000496
isFraud	0.025495	0.039389	0.009226	-0.008322	-0.005657	0.000496	1.000000

Рисунок 2.13 Кореляція між відкоригованими даними ознак

Можемо побачити, що тепер newbalanceDest та oldbalanceDest мають сильний зв’язок між собою.

Побудуємо стовпчасту діаграму, яка показує кореляцію між змінною isFraud (цільова змінна) та всіма іншими числовими змінними в наборі даних.

Зверху вниз, змінні розташовані у порядку спадання їх кореляції зі змінною isFraud:

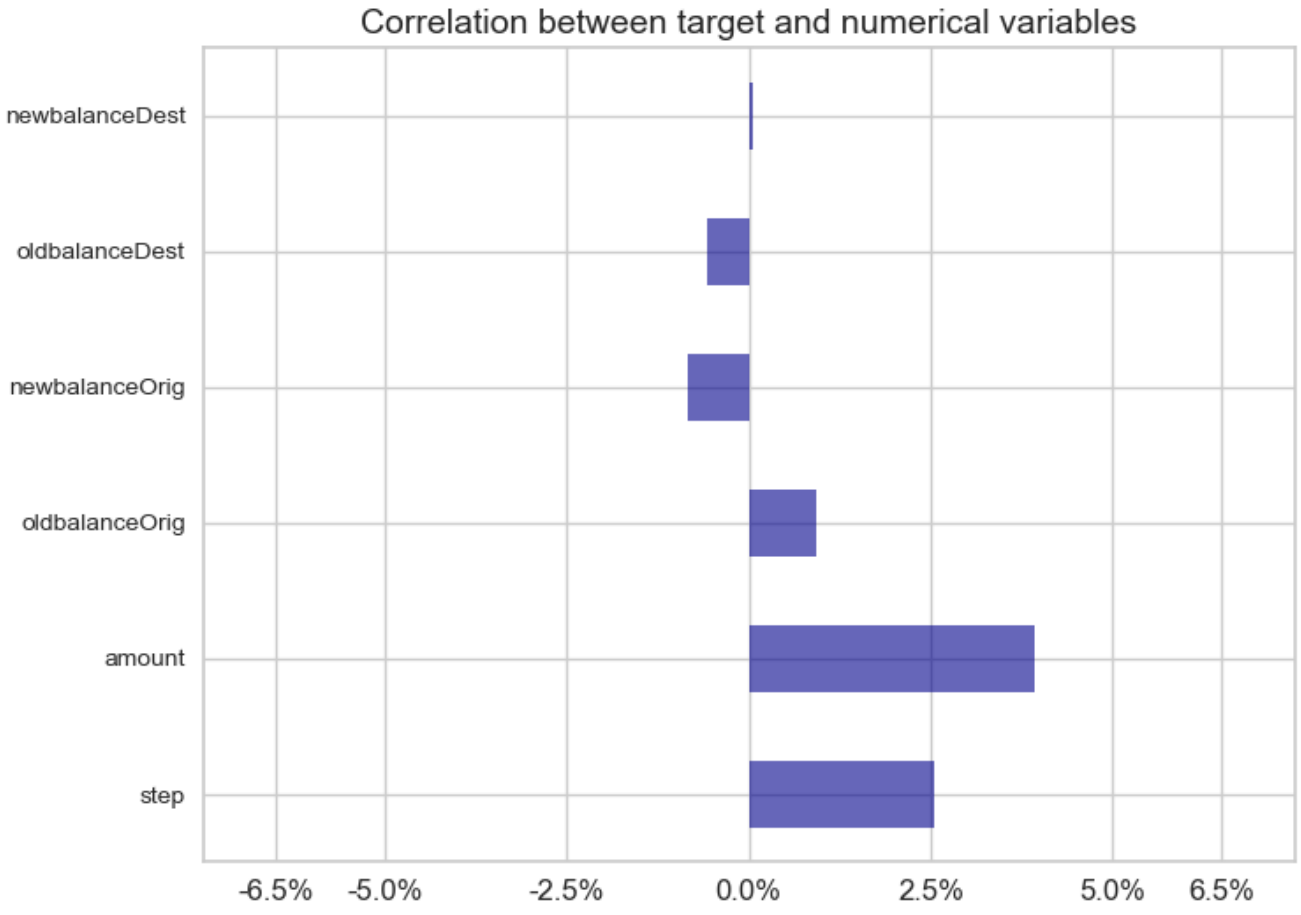

Рисунок 2.14 Кореляція між isFraud та числовими змінними

Від'ємне значення кореляції означає, що змінні рухаються в протилежних напрямках, тоді як додатне значення означає, що змінні рухаються в одному напрямку.

Графік чітко відображає, що amount найбільш зв’язаний з isFraud.

Побудуємо стовпчикову діаграму, яка показує кількість унікальних значень для кожної категоріальної ознаки з набору даних, де:

y - кількість унікальних значень кожної категоріальної ознаки з набору даних train[cat_columns].

x - імена категоріальних ознак з train[cat_columns].

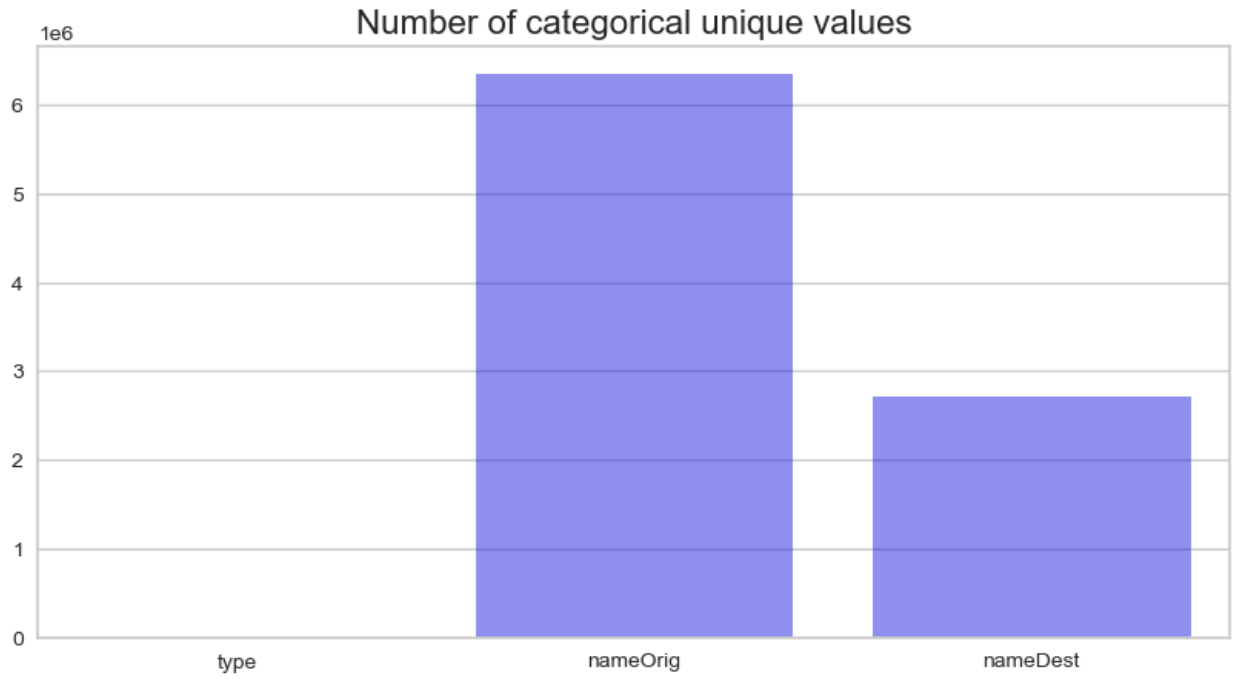

Рисунок 2.15 Кількість унікальних значень у датасеті

Інформація про унікальні значення в категоріальних стовпцях набору даних може бути корисною з декількох причин:

1. Розуміння розподілу категорій: Знання кількості унікальних значень допомагає нам отримати уявлення про розподіл категорій у стовпці. Це може бути важливою інформацією для подальшого аналізу та врахування особливостей розподілу у моделюванні.

2. Виявлення високовимірних стовпців: Якщо категоріальний стовпець має велику кількість унікальних значень, це може свідчити про високу вимірність цього стовпця. Високовимірні стовпці можуть бути складними для обробки та аналізу, а також можуть призводити до перенавчання моделей. Знання про кількість унікальних значень допоможе нам виявити такі стовпці та прийняти рішення щодо їх обробки або включення в модель.

3. Кодування категоріальних змінних: при роботі з категоріальними змінними для моделювання часто потрібно закодувати їх у числовий формат. Знання про кількість унікальних значень допомагає визначити, який метод кодування буде найкращим в конкретному випадку. Наприклад, якщо категоріальний стовпець має декілька унікальних значень, можна розглянути застосування методу "one-hot encoding", тоді як для стовпця з великою кількістю унікальних значень може бути доцільним використовувати кодування за допомогою "label encoding" або "target encoding".

Як бачимо, велика кількість унікальних категоріальних значень у цьому датасеті потребує кодування для того, щоб наша модель класифікації працювала без помилок.

Обчислемо кількість записів з кожним значенням ознаки "isFraud" у наборі даних.:

```
0    6343476
1       7717
Name: isFraud, dtype: int64
```

Рисунок 2.16 Кількість шахрайських та легітимних транзакцій

Це означає, що в наборі даних є 6343476 транзакцій, які не є шахрайськими (значення 0) та 7717 транзакції, які є шахрайськими (значення 1) за ознакою "isFraud". Тому можемо підтвердити, що наш набір даних є незбалансованим.

Відобразимо на графіку цю незбалансованість для розуміння степені незбалансованості:

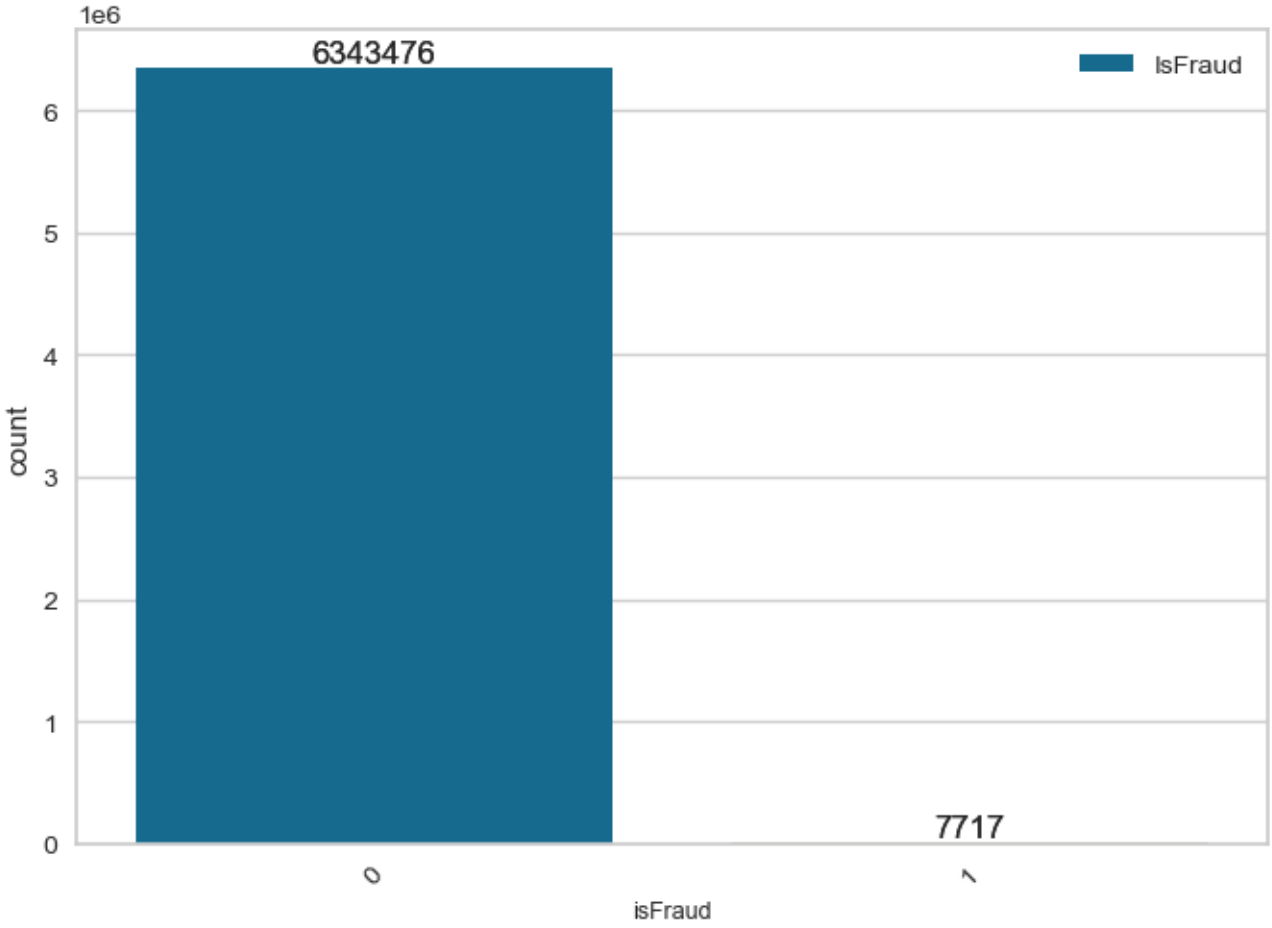

Рисунок 2.17 Графік незбалансованості даних шахрайств

Побудуємо графіки скрипичних діаграм (violinplot) для візуалізації зв'язку між числовими ознаками та цільовою змінною 'isFraud'.

Побудова цих графіків має такі цілі:

1. Візуалізація розподілу значень числових ознак залежно від класу 'IsFraud' (шахрайство або не шахрайство). Це допомагає зрозуміти, які значення ознак мають різниці або схожості між класами. Графіки скрипки дозволяють побачити щільність значень ознак, медіану, міжквартильний розмах та діапазон значень для кожного класу.

2. Виявлення потенційних розбіжностей у розподілі ознак між класами 'IsFraud'. Якщо графіки скрипки для певної ознаки мають різні форми або розташування медіани, це може вказувати на важливість цієї ознаки для класифікації шахрайства.

3. Виявлення викидів або незвичайних значень у ознаках для кожного класу. Графіки скрипки можуть допомогти виявити аномалії або екстремальні значення у розподілі ознак, які можуть бути пов'язані з шахрайською активністю.

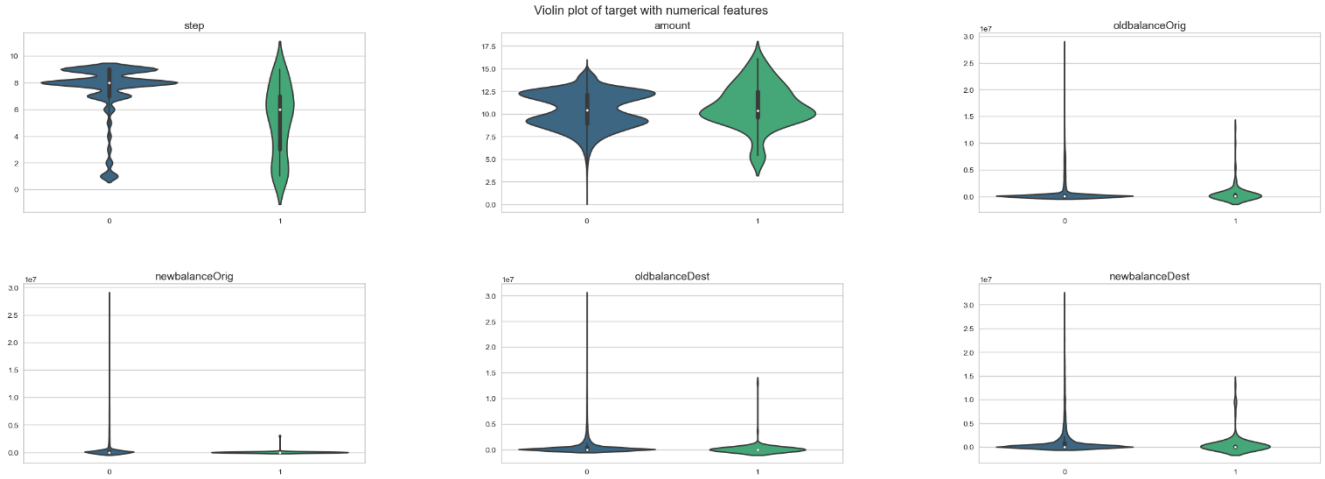

Рисунок 2.18 Графіки скрипичних діаграм

Побудова цих графіків має такі цілі:

1. Візуалізація розподілу значень числових ознак залежно від класу 'IsFraud' (шахрайство або не шахрайство). Це допомагає зрозуміти, які значення ознак мають різниці або схожості між класами. Графіки скрипки дозволяють побачити щільність значень ознак, медіану, міжквартильний розмах та діапазон значень для кожного класу.

2. Виявлення потенційних розбіжностей у розподілі ознак між класами 'IsFraud'. Якщо графіки скрипки для певної ознаки мають різні форми або розташування медіани, це може вказувати на важливість цієї ознаки для класифікації шахрайства.

3. Виявлення викидів або незвичайних значень у ознаках для кожного класу. Графіки скрипки можуть допомогти виявити аномалії або екстремальні значення у розподілі ознак, які можуть бути пов'язані з шахрайською активністю.

Можемо побачити, що найбільший розмах ширини значень isFraud має newbalanceOrig, також бачимо підтвердження що ознаки newbalanceDest та oldbalanceDest мають сильний зв’язок між собою, бо їх розмах майже ідентичний.

Створимо графік із 6 підграфіками, кожен з яких показує KDE (ядерну оцінку щільності) числової ознаки у датафреймі.

Ядерна оцінка щільності є корисним інструментом для візуалізації та аналізу розподілу даних, виявлення відмінностей між різними групами та оцінки ймовірності.

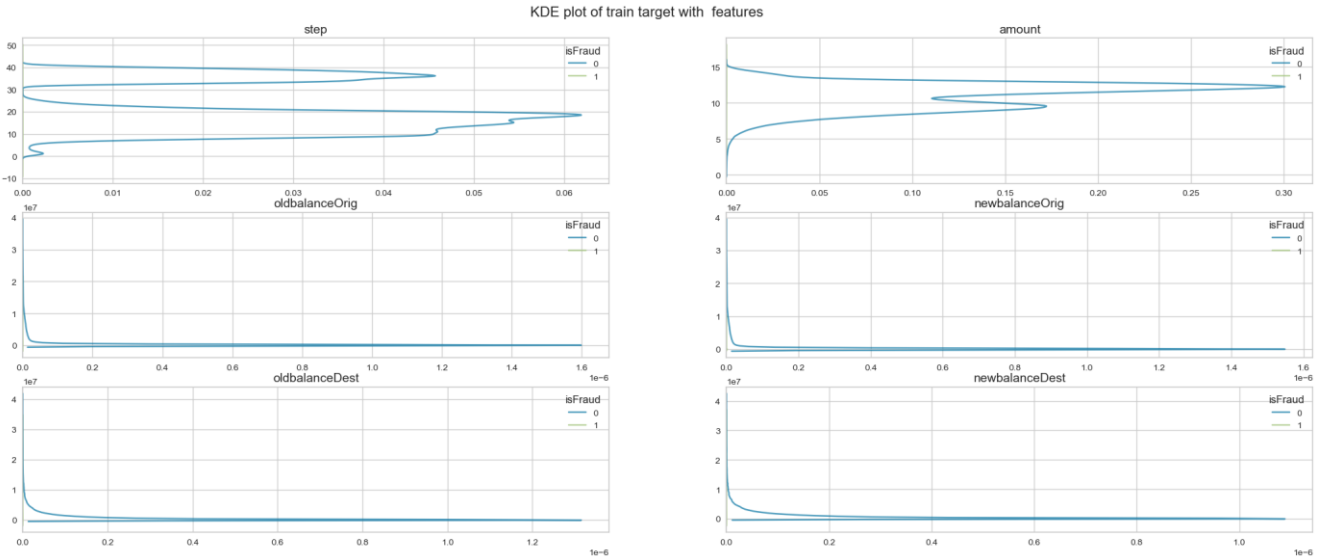

Рисунок 2.19 Ядерна оцінка

Трансформуємо дані з використанням алгоритму TSNE (t-distributed stochastic neighbor embedding) для зменшення кількості вимірів даних і візуалізації їх в двовимірному просторі. Спочатку з набору даних вилучаються нечислові ознаки, які не можуть бути використані для tsne-преобразування. Після цього з застосуванням RobustScaler числові дані нормалізуються і підготовлюються для використання в tsne. Нарешті, виконується само tsne-преобразування, яке зменшує кількість вимірів з метою візуалізації даних в двовимірному просторі. Результатом є numpy масив з двома стовпцями, що представляють нові координати кожної точки даних в двовимірному просторі.

Потім виконаємо візуалізація точок на площині x-y залежно від значення стовпця isFraud:

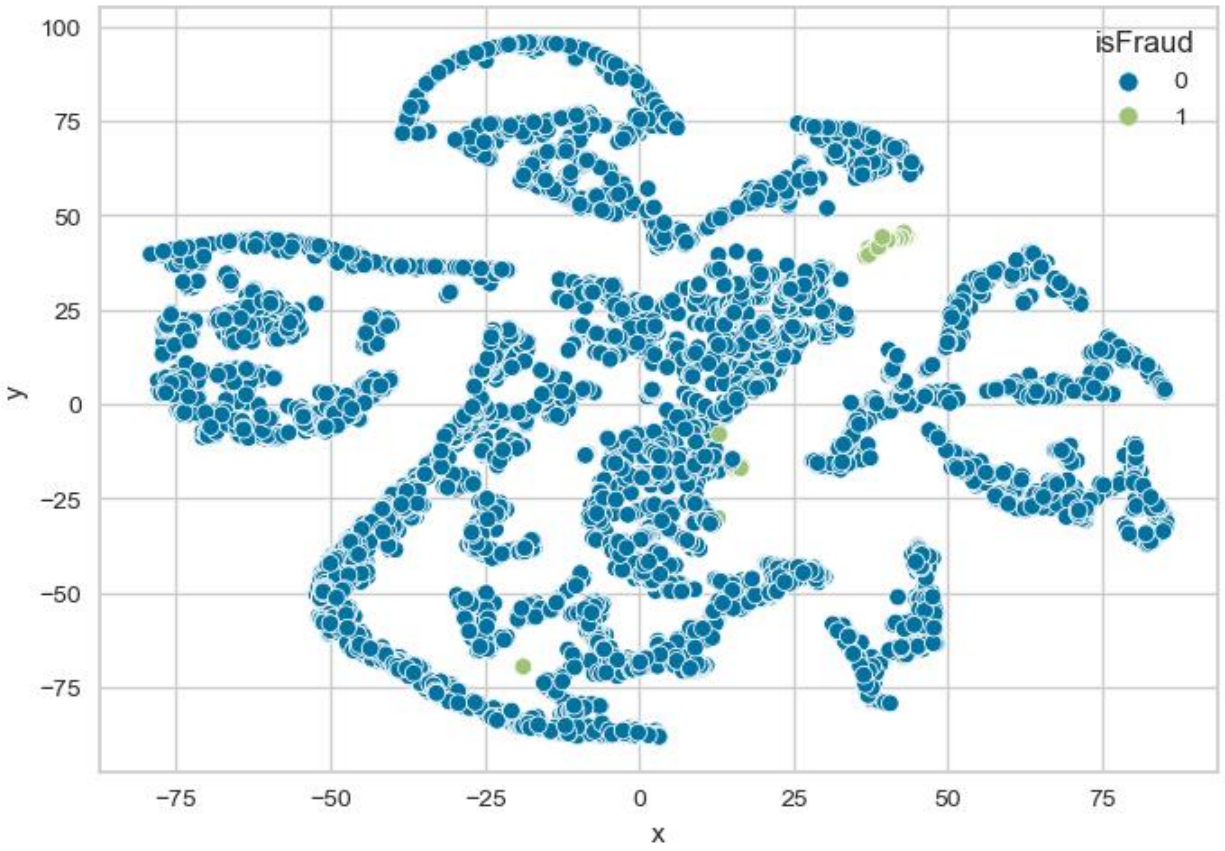

Рисунок 2.20 Візуалізація точок шахрайства

Як ми бачимо, нелінійне перетворення дає нам деякий кластер шахрайства, добре відокремлений від нешахрайських даних.

Далі розіб’ємо навчальний датасет на дві частини - навчальний та тестовий. Розмір тестового датасету встановлюється на 10% від загального розміру. Крім того, цей метод робить збалансований поділ даних, використовуючи параметр stratify=y. Це означає, що кожен окремий клас має представника як у навчальному, так і в тестовому датасетах.

```
635120 rows in test set vs. 5716073 in training set. 9 Features.
```

Рисунок 2.21 Результат розділення

На цьому етапі розвідковий аналіз даних завершено. У наступному розділі приступимо до програмування та тесту моделей прогнозування шахрайства.

2.4 Побудова моделей прогнозування

Перед тим як будувати моделі, спочатку кластеризуємо навчальний датасет за допомогою двох методів: K-means та Fuzzy Clustering. Ці методи кластерізації можуть поліпшити якість наших моделей, ізолюючи більшість шахрайств у одному кластері.

2.4.1 Модель прогнозування з кластерізацією K-means

Перед тим як кластерізувати дані, спочатку закодуємо категоріальні ознаки type, nameOrig та nameDest за допомогою Label Encoder.

```
step              int64
type              int32
amount          float64
nameOrig          int32
oldbalanceOrig  float64
newbalanceOrig  float64
nameDest          int32
oldbalanceDest  float64
newbalanceDest  float64
dtype: object
step              int64
type              int32
amount          float64
nameOrig          int32
oldbalanceOrig  float64
newbalanceOrig  float64
nameDest          int32
oldbalanceDest  float64
newbalanceDest  float64
dtype: object
```

Рисунок 2.4.1.1 Результати кодування

Далі нам потрібно нормалізувати дані за допомогою алгоритму Standard Scaler.

Після цього нам потрібно знати, на скільки кластерів потрібно розділити дані. Для цього реалізуємо метод "Лікоть" (Elbow Method) для визначення оптимальної кількості кластерів у алгоритмі K-means. Він обчислює значення інерції (суми квадратів відстаней) для різних кількостей кластерів і показує графік "крива лікоть". Оптимальне число кластерів визначається за допомогою знаходження точки "лікоть" на графіку.

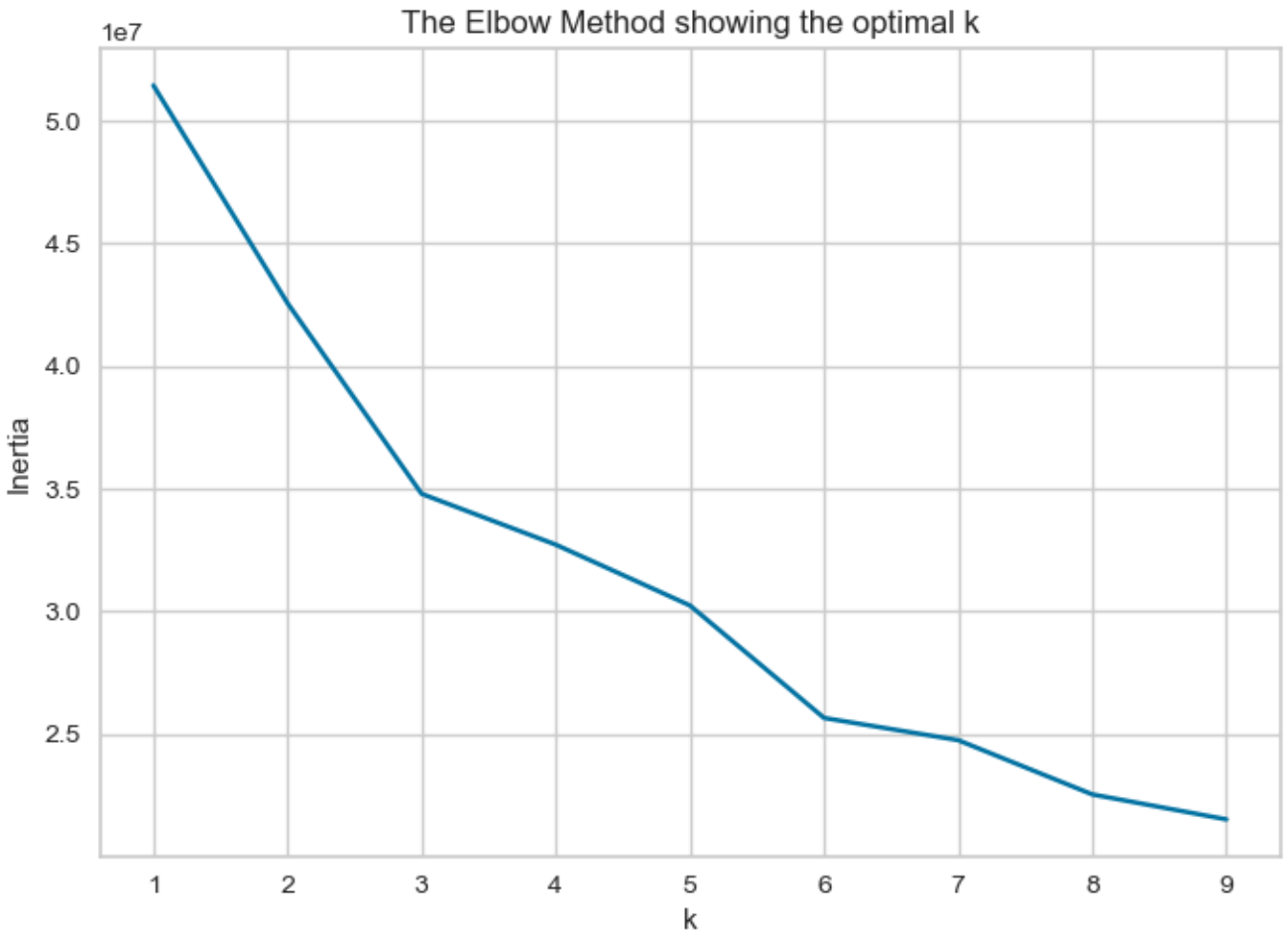

Рисунок 2.4.1.2 Оптимальна кількість кластерів

Після цього реалізуємо метод K-means.

Виконуємо наступні кроки:

1. Застосовується кодування міток класів у вихідних даних за допомогою LabelEncoder. Вихідні мітки класів у y_train перетворюються на числові значення.

2. Використовується отримане оптимальне число кластерів (елбоу індекс) для створення інстанції алгоритму кластеризації K-means (MiniBatchKMeans).

3. Алгоритм K-means навчається на навчальних даних (X_train_scaled) і призначає кластери кожній точці даних у навчальному наборі.

Отримані кластерні мітки для навчального набору зберігаються в змінній cluster_labels_train.

Далі перевіримо в якому кластері найбільша кількість шахрайтсв:

Таблиця 2.4.1.1 Процент шахрайств у кластерах

isFraud	0	1	All
cluster			
0	0.545078	0.518790	0.545046
1	0.413195	0.468539	0.413263
2	0.041727	0.012671	0.041692
All	1.000000	1.000000	1.000000

Як бачимо, таблиця показує, що у кластері №0 найбільша кількість шахрайств.

Перевіримо також який процент шахрайств є у кожному з кластерів:

```
Fraud percentage in Cluster 0: 56.21761658031088
Fraud percentage in Cluster 1: 42.35751295336787
Fraud percentage in Cluster 2: 1.4248704663212435
100.0
```

Рисунок 2.4.1.3 Процент шахрайств у кожному з кластерів

Бачимо, що що кластер №0 дійсно є лідером у проценті шахрайств. Також ми маємо біль детальну інформацію щодо інших кластерів, за допомогою якої ми можемо зрозуміти, хто з останніх двох кластерів має більшу кількість шахрайств, а саме кластер № 1. У такому випадку, можемо провести додатковий експеримент з підвищення якості моделі прогнозування – скомбінувати кластер №0 та №1 та зробити на цих даних модель прогнозування.

Перевіримо точну кількість для перевірки, що програма правильно обчислила процент шахрайств:

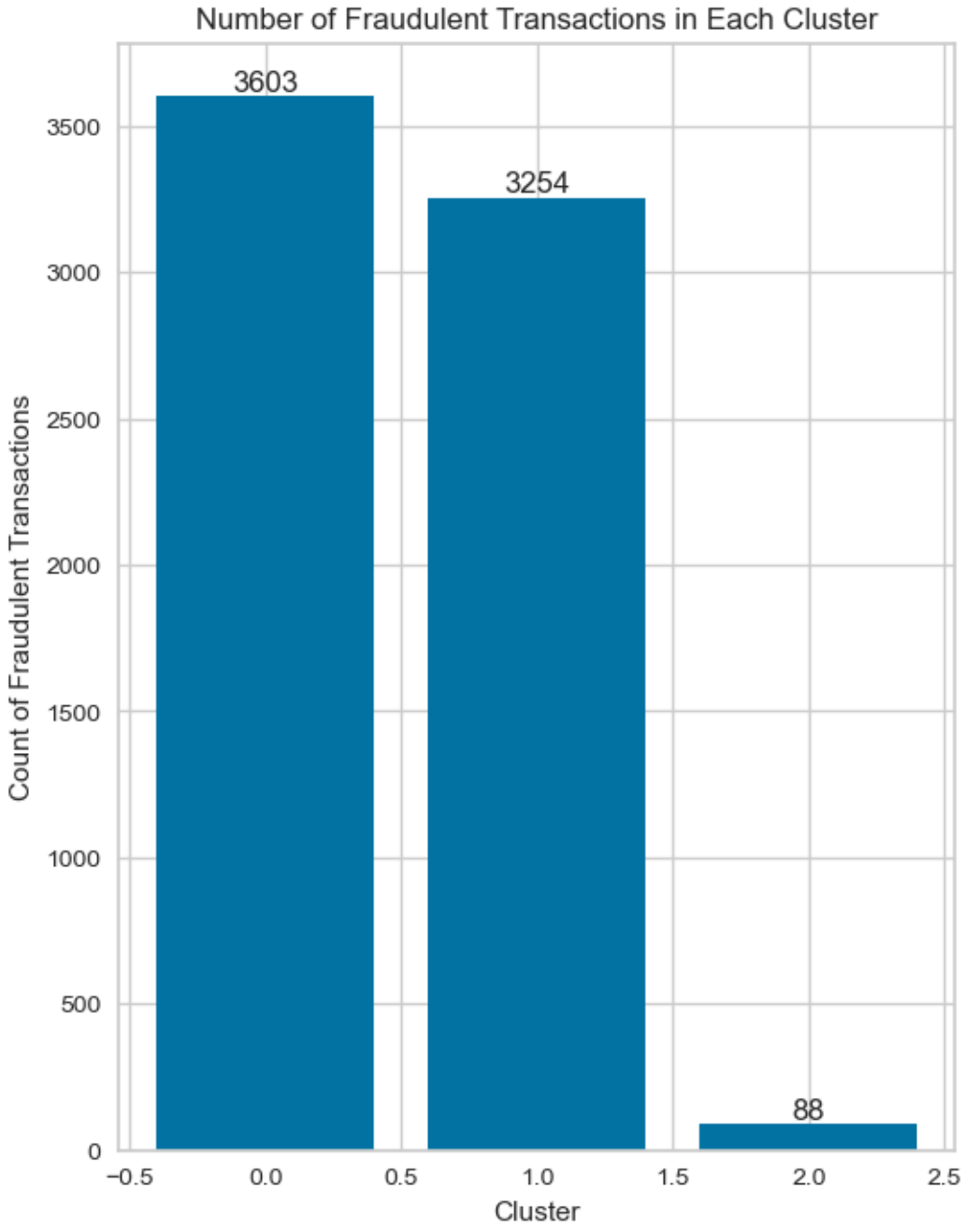

Рисунок 2.4.1.4 Кількість шахрайських даних у кожному кластері

Також візуалізуємо кількість даних у кожному з кластерів:

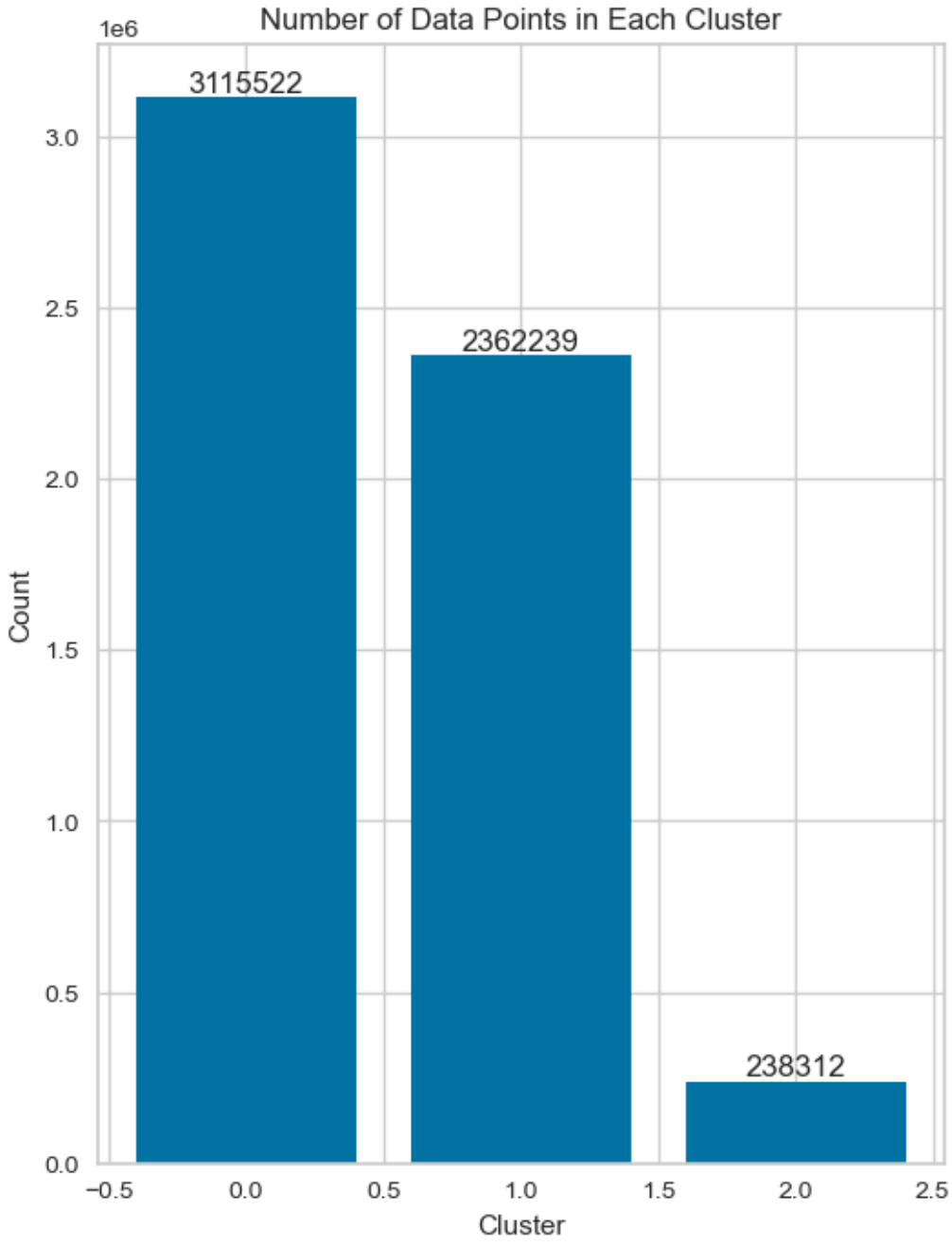

Рисунок 2.4.1.5 Кількість даних у кожному кластері

Тепер виконуємо наступні дії:

1. Створюється новий навчальний набір, який включає лише точки даних, що належать до кластера 0. Відповідні мітки класів також відображаються у відповідній змінній.

2. Отримуються кластерні мітки для кожної точки даних у тестовому наборі.

3. Створюється новий тестовий набір, який включає лише точки даних, що належать до кластера 0. Відповідні мітки класів також відображаються у відповідній змінній.

Отримані набори X_train_cluster0, y_train_cluster0, X_test_cluster0 і y_test_cluster0 містять тільки дані, які відповідають кластеру 0.

Тепер можемо побудувати модель прогнозування на основі даних кластеру №0. Виконуємо наступні дії:

1. Створюється об'єкт класифікатора XGBoost (xgb_model_clust_0).

2. Проводиться навчання моделі на навчальних даних з використанням валідаційного набору, що вказаний (eval_set=[(X_test_cluster0, y_test_cluster0)]). early_stopping_rounds вказує на кількість ітерацій, протягом яких буде очікуватися покращення метрики якості на валідаційному наборі перед зупинкою навчання. verbose=1 вказує на виведення інформації про процес навчання.

3. Здійснюються передбачення моделі на тестовому наборі, що відповідає кластеру 0 (y_pred = xgb_model_clust_0.predict(X_test_cluster0)).

4. Обчислюється точність моделі (accuracy_score), логлосс (log_loss), матриця плутанини (confusion_matrix), звіт про класифікацію (classification_report), F1-оцінка (f1_score) і AUC-ROC (roc_auc_score).

```
[97]    validation_0-logloss:0.00110
[98]    validation_0-logloss:0.00110
[99]    validation_0-logloss:0.00110
Accuracy: 0.9996113295982587
Logloss: 0.0010964476809411875
[[346889     15]
 [   120    314]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    346904
           1       0.95      0.72      0.82       434

    accuracy                           1.00    347338
   macro avg       0.98      0.86      0.91    347338
weighted avg       1.00      1.00      1.00    347338

F1-score: 0.8230668414154653
AUC-ROC: 0.8617295322596055
```

Рисунок 2.4.1.6 Результати моделі прогнозування на основі даних кластеру №0

Результати оцінки моделі XGBoost на тестових даних, що відповідають кластеру 0, є наступними:

- Точність (Accuracy): 0.9996113295982587, що означає, що модель правильно класифікує 99.96% прикладів.

- Логлосс (Logloss): 0.0010964476809411875, який є дуже низьким значенням. Чим менше значення логлосс, тим краще модель прогнозує ймовірності класів.

- Confusion Matrix:

- True negatives (TN): 346889 - кількість правильно класифікованих негативних прикладів.

- False positives (FP): 15 - кількість негативних прикладів, які були неправильно класифіковані як позитивні.

- False negatives (FN): 120 - кількість позитивних прикладів, які були неправильно класифіковані як негативні.

- True positives (TP): 314 - кількість правильно класифікованих позитивних прикладів.

- F1-оцінка становить 0.8230668414154653 для позитивного класу.

- AUC-ROC: 0.8617295322596055 вказує на те, що модель XGBoost має хорошу дискримінаційну здатність. Це свідчить про те, що модель має високу ймовірність ранжування вибраного випадкового позитивного екземпляру вище, ніж вибраного випадкового негативного екземпляру при встановленні різних порогових значень.

Загалом, модель показує дуже високу точність та хорошу здатність до виявлення фроду (позитивного класу). Проте, такий F1-score буде найменшим з усіх інших моделей, що вказує на гіршу якість цієї моделі.

Далі побудуємо дві нові моделі, на основі лише тренувальних даних та на даних, де кластери є додатковою ознакою у тренувальних та тестових датасетах для порівняння якості моделей.

Модель на основі тренувальних та тестових даних:

```
Accuracy: 0.9997732711928454
Logloss: 0.0007949820151272467
[[634325     23]
 [   121    651]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    634348
           1       0.97      0.84      0.90       772

    accuracy                           1.00    635120
   macro avg       0.98      0.92      0.95    635120
weighted avg       1.00      1.00      1.00    635120

F1-score: 0.900414937759336
AUC-ROC: 0.9216139955019214
```

Рисунок 2.4.1.7 Результат прогнозування на навчальних та тестових даних

- Точність (Accuracy): 0.9997732711928454, що означає, що модель правильно класифікує 99.98% прикладів.
- Логлосс (Logloss): 0.0007949820151272467, який є дуже низьким значенням.
- Confusion Matrix:
- True negatives (TN): 634325 - кількість правильно класифікованих негативних прикладів.
- False positives (FP): 23 - кількість негативних прикладів, які були неправильно класифіковані як позитивні.
- False negatives (FN): 121- кількість позитивних прикладів, які були неправильно класифіковані як негативні.
- True positives (TP): 651 - кількість правильно класифікованих позитивних прикладів.
- F1-оцінка становить 0.9004 для позитивного класу.
- AUC-ROC: 0.9216 вказує на те, що модель XGBoost має хорошу дискримінаційну здатність.

Можемо відразу побачити, що F1-score значно більший за моделі кластера №0.

Тепер побудуємо модель з номерами кластерів у вигляді додаткової ознаки:

```
[97]    validation_0-logloss:0.00078
[98]    validation_0-logloss:0.00077
[99]    validation_0-logloss:0.00077
Accuracy: 0.9997748456984507
[[634327     21]
 [   122    650]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    634348
           1       0.97      0.84      0.90       772

    accuracy                           1.00    635120
   macro avg       0.98      0.92      0.95    635120
weighted avg       1.00      1.00      1.00    635120

F1-score: 0.9009009009009008
AUC-ROC: 0.9209679035299136
```

Рисунок 2.4.1.8 Результат прогнозування з номерами кластерів

Можемо відразу побачити, що f1-score став краще моделі прогнозування без кластерізації на 0.005%, що є незначним покращенням якості, але найточнішу різницю в якості покаже кросс-валідація.

А тепер, побачивши результати моделей XGBoost з різними даними у вигляди кластера № 0 та міток кластерів, побудуємо модель з комбінованими даними кластерів №0 та № 1. Для цього виконуємо фільтрацію навчального та тестового наборів даних з урахуванням кластерних міток. Фільтрація проводиться для кластерів з мітками 0 і 1, при цьому зберігаються всі стовпці ознак.

1. Для навчального набору даних:
    - X_train_cluster01 створюється шляхом вибору лише тих рядків з X_train_scaled, де кластерні мітки дорівнюють 0 або 1. Це виконується за допомогою функції np.logical_or, яка створює булевий масив, що містить True, якщо кластерна мітка дорівнює 0 або 1, та False в протилежному випадку.
    - y_train_cluster01 містить відповідні категоріальні мітки з y_train_encoded для відфільтрованих рядків у X_train_cluster01. Використовується та сама логіка фільтрації з np.logical_or, щоб вибрати відповідні мітки.
2. Для тестового набору даних:
    - X_test_cluster01 створюється шляхом вибору лише тих рядків з X_test_scaled, де кластерні мітки дорівнюють 0 або 1. Аналогічно до навчального набору даних, використовується np.logical_or для фільтрації рядків.

- y_test_cluster01 містить відповідні категоріальні мітки з y_test_encoded для відфільтрованих рядків у X_test_cluster01. Знову ж таки, використовується та сама логіка фільтрації з np.logical_or, щоб вибрати відповідні мітки.

Отже, після виконання цього коду у нас будуть збережені відфільтровані навчальний і тестовий набори даних, які містять лише рядки, що належать до кластерів з мітками 0 або 1, разом з відповідними категоріальними мітками.

Тепер побудуємо модель прогнозування XGBoost на цих даних:

```
[37]    validation_0-logloss:0.00189
[38]    validation_0-logloss:0.00186
[39]    validation_0-logloss:0.00163
Accuracy: 0.9996449762495685
Logloss: 0.0016119901138699122
[[607610     39]
 [   177    584]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    607649
           1       0.94      0.77      0.84       761

    accuracy                           1.00    608410
   macro avg       0.97      0.88      0.92    608410
weighted avg       1.00      1.00      1.00    608410

F1-score: 0.8439306358381503
AUC-ROC: 0.8836735595653422
```

Рисунок 2.4.1.9 Модель прогнозування на даних з кластера №0 та №1

Бачимо, що маємо один з гірших результатів усіх інших моделей, причиною цьому може бути перенавчання моделі. Можемо змінити деякі параметри моделі, щоб подивитись, чи можемо ми покращити цей результат:

```
[430]   validation_0-logloss:0.00068
[431]   validation_0-logloss:0.00068
[432]   validation_0-logloss:0.00068
Accuracy: 0.9997731792705576
Logloss: 0.0006820731487486438
[[607632     17]
 [   121    640]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    607649
           1       0.97      0.84      0.90       761

    accuracy                           1.00    608410
   macro avg       0.99      0.92      0.95    608410
weighted avg       1.00      1.00      1.00    608410

F1-score: 0.9026798307475318
AUC-ROC: 0.9204853546311139
```

Рисунок 2.4.1.10 Глибина 5, кількість дерев – 500, learning_rate = 0.1, subsample =1, colsample_bytree = 1, tree_method = 'auto'

Можемо побачити, що зміна глибини з стандартних 3 до 5, та кількість дерев з 100 до 500 покращили якість моделі.

Також для порівняння самої моделі XGBoost, зробимо модель прогнозування Random Forest на тренувальних та тестових даних:

```
Accuracy: 0.999685098878952
Logloss: 0.03819711016605843
[[634336     12]
 [   188    584]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    634348
           1       0.98      0.76      0.85       772

    accuracy                           1.00    635120
   macro avg       0.99      0.88      0.93    635120
weighted avg       1.00      1.00      1.00    635120

F1-score: 0.8538011695906432
AUC-ROC: 0.8782288834382632
```

Рисунок 2.4.1.11 Результат Random Forest

Результат цієї моделі показує один з гірших результатів. Але на цьому прикладі можемо побачити, що результат точності у всіх моделей майже 99,99%, що свідчить про те, що точність не є значимою мірою якості моделі.

Тепер розглянемо, які ознаки були найбільш значущі у процесі прогнозування моделей:

Кластер №0:

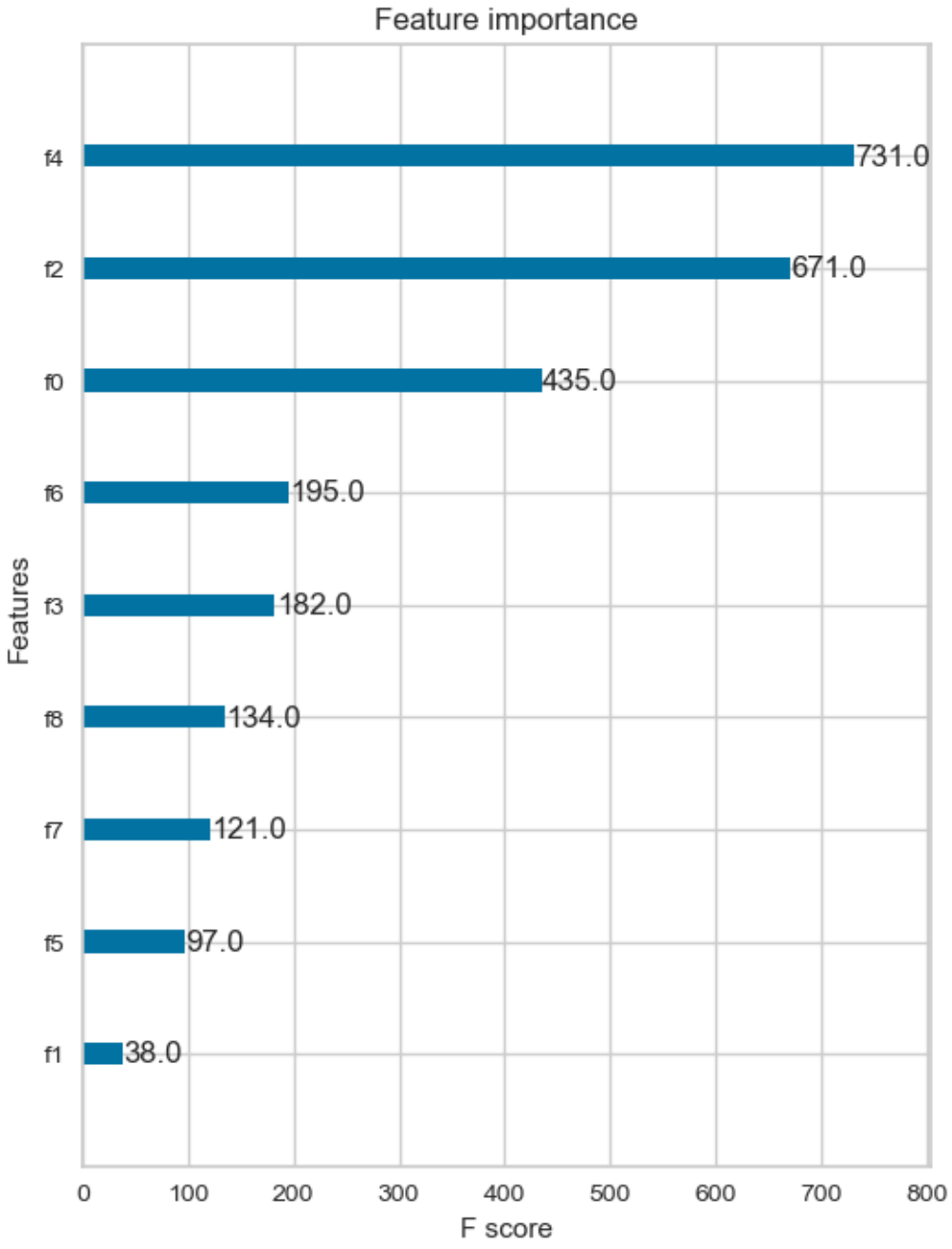

Рисунок 2.4.1.12 Feature Importance моделі кластера №0

Бачимо, що найбільш значущими ознаки були f2 (amount) та f4 (oldbalanceOrig), що свідчить про сильний зв’язок цих ознак з шахрайствами, що буде корисною інформацією для рішення задач про виявлення фроду.

Модель на даних тренувального та тестового датасетах:

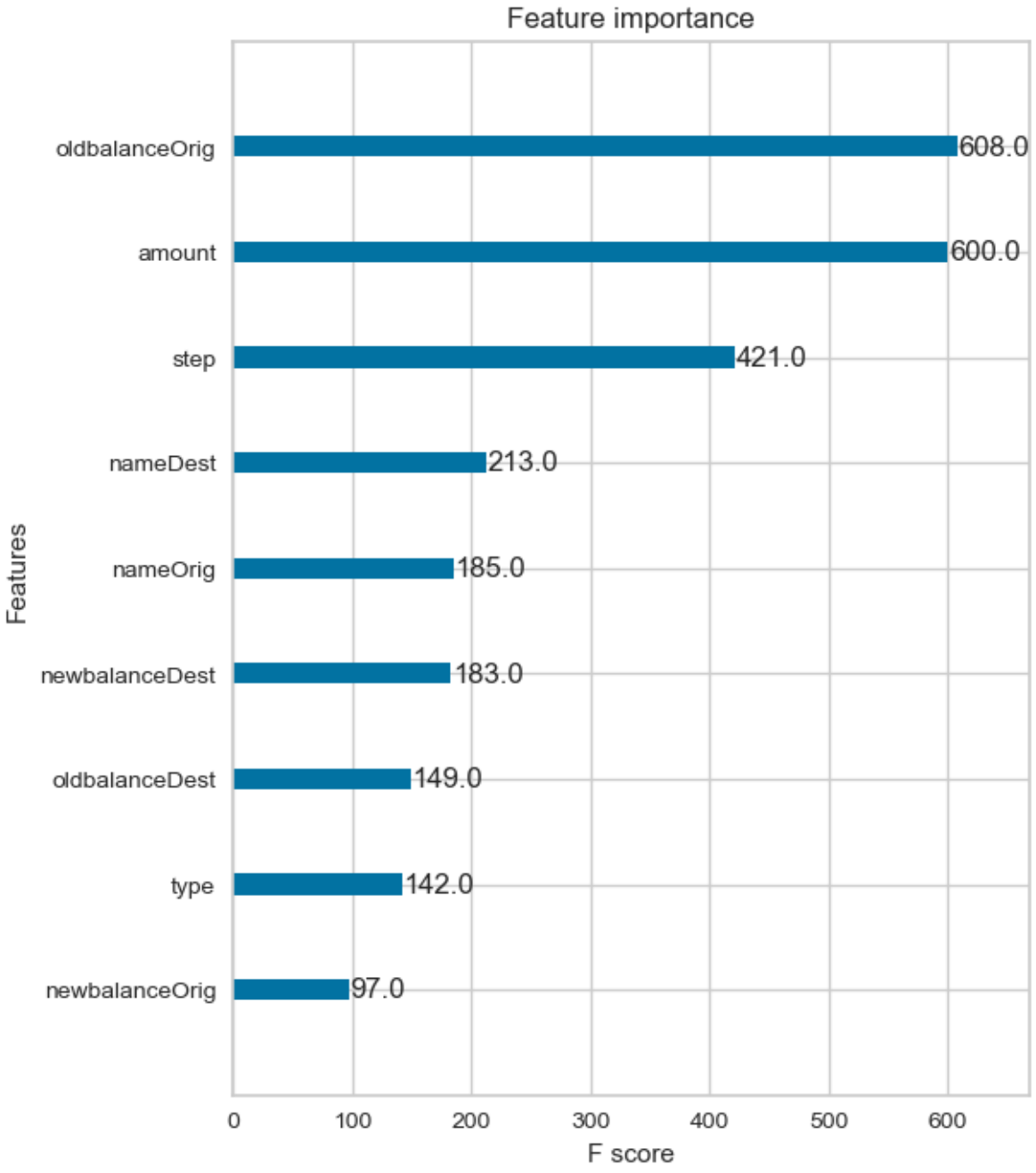

Рисунок 2.4.1.13 Feature Importance моделі на тренувальних та тестових даних

Бачимо майже ідентичні зв’язки, але в цьому методі oldbalanceOrig меньш сильний зв’язок.

Назви кластерів у вигляді додаткової ознаки:

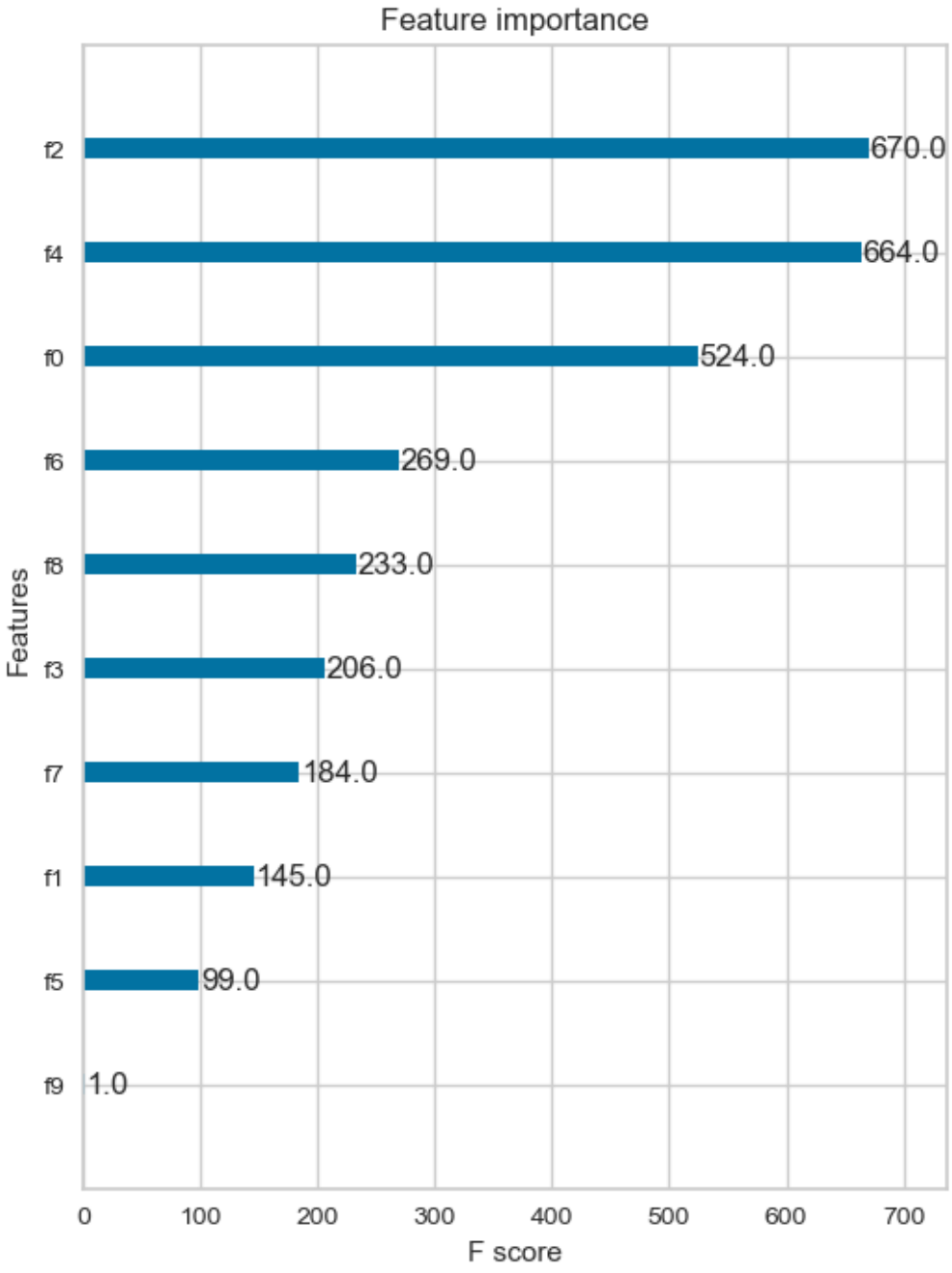

Рисунок 2.4.1.14 Feature Importance моделі з додатковою ознакою у вигляді назв кластерів.

Тут можемо побачити, що кластери не вплинули на результат моделі.

Виконаємо кросс-валідацію для моделей оригінальних наборів даних, модель К-середніх, модель комбінованих даних та random Forest та порівняємо їх за допомогою візуалізації.

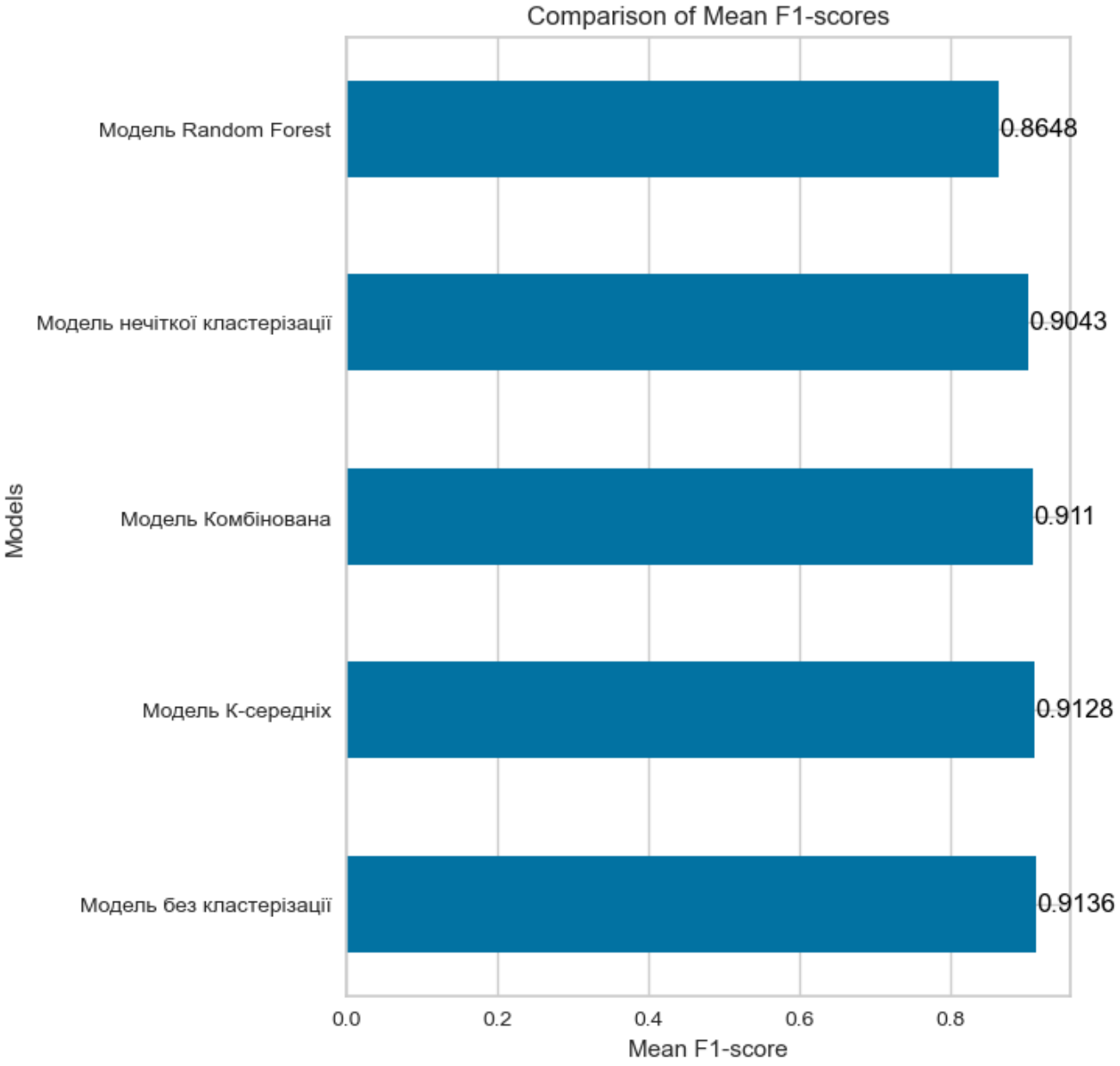

Рисунок 2.4.1.15 Візуалізація кросс-валідації моделей прогнозування

Кросс-валідація показує, що найкращою моделлю якості є модель без кластеризації, але їх різниця є дуже малою, якщо порівнювати з моделлю комбінованою та К-середніх, а також гіршу якість Random Forest.

Але це є результатами середнього значення між результатами кросс-валідації, тому подивимося на масив деяких моделей прогнозування, а саме модель без кластеризації та комбіновану:

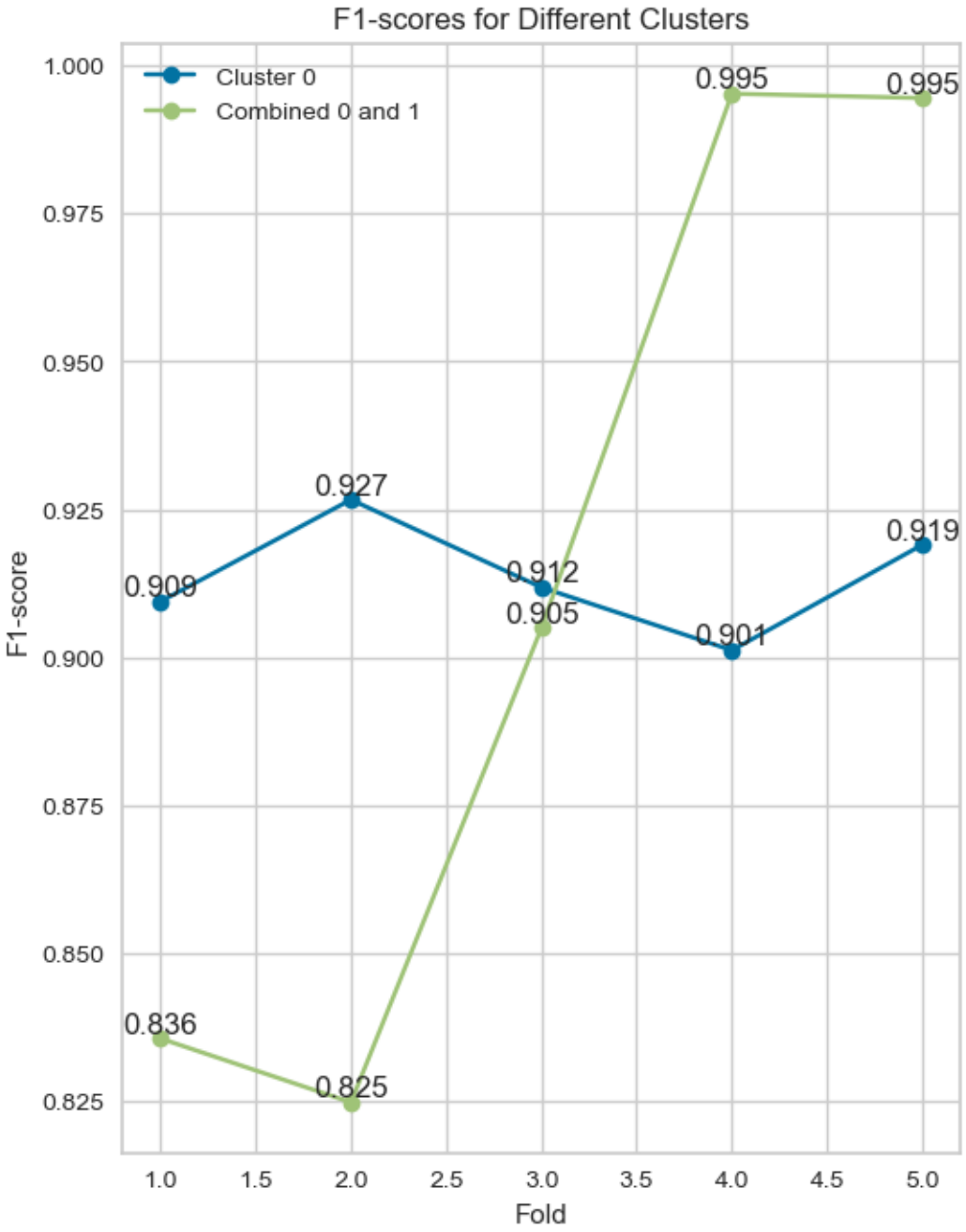

Рисунок 2.4.1.16 Масиви кросс-фалідацій для моделей без кластеризації та комбінованої

Ці оцінки представляють ефективність моделі для кожного окремого тестового набору даних. Вони вказують на те, наскільки добре модель працює для кожного конкретного випадку під час кросс-валідації. Більші значення F1-оцінки вказують на кращу ефективність моделі при класифікації.

Бачимо, що комбінований метод дає у останніх тестах найбільший результат, що свідчить про вдосконалення якості моделі при подальших тестах, але при узагальненні, кращою стає модель без кластеризації за рахунок збалансованості її масиву.

Далі розглянемо результати нечіткої кластерізації.

2.4.2 Метод Fuzzy Clustering

Спочатку обчислимо оптимальну кількість кластерів для кластерізації, для цього виконаємо наступні дії:

1. Створюється вибірка даних за допомогою функції make_blobs. Ця функція генерує зразки даних у формі кластерів, де кожен кластер представляється як гаусіанський розподіл.

2. Визначається діапазон значень k, які будуть використовуватись для тестування алгоритму кластеризації. У даному випадку, діапазон складається з чисел від 2 до 9.

3. Створюється пустий список f_indices, який буде містити значення F-індексу для кожного значення k.

4. Виконується цикл, в якому обчислюється F-індекс для кожного значення k. Для кожного k, використовуючи алгоритм Fuzzy C-means, обчислюються центри кластерів cntr, матриця приналежності u, а також значення F-індексу f. Значення F-індексу додається до списку f_indices.

5. Побудова графіка залежності F-індексу від значення k. На графіку вісь X представляє значення k, а вісь Y - значення F-індексу.

6. Визначення оптимального значення k як індексу з найбільшим значенням F-індексу. Оскільки діапазон k починається з 2, то до індексу додається 2.

7. Виведення на екран оптимального значення k.

Цей код виконує пошук оптимального значення k для алгоритму Fuzzy C-means, використовуючи F-індекс як метрику оцінки якості кластеризації. Це дозволяє визначити кількість оптимальних кластерів в даних.

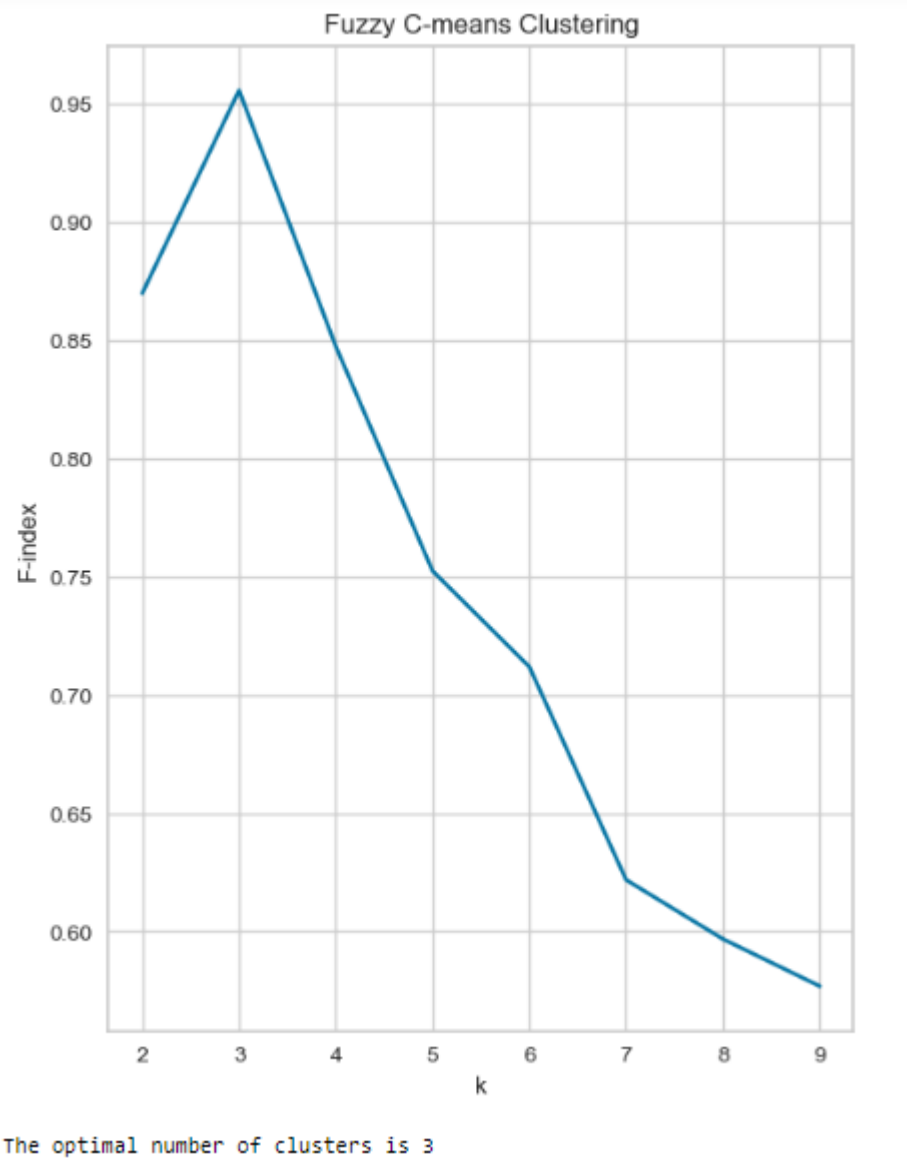

Рисунок 2.4.2.1 Оптимальна кількість кластерів

Далі проведемо наступні операції:

1. Імпорт бібліотеки LabelEncoder з модулю sklearn.preprocessing. LabelEncoder використовується для перетворення категоріальних міток в числовий формат.

2. Створення екземпляра LabelEncoder - об'єкту, який виконує перетворення.

3. Застосування методу fit_transform до категоріальних міток у y_train. Метод fit_transform спочатку навчає LabelEncoder на унікальних значеннях міток у y_train і потім застосовує перетворення до y_train, замінюючи категоріальні мітки числовими значеннями.

4. Застосування методу transform до категоріальних міток у y_test. Метод transform застосовує перетворення, навчене на y_train, до y_test, замінюючи категоріальні мітки числовими значеннями.

5. Імпорт бібліотеки skfuzzy з модулю skfuzzy. Бібліотека skfuzzy використовується для роботи з нечіткими множинами та нечітким кластеруванням.

6. Встановлення кількості кластерів n_clusters на значення 3.

7. Застосування алгоритму нечіткого кластерування C-середніх (fuzzy C-means) до X_train. Метод fuzz.cluster.cmeans приймає матрицю X_train (транспоновану) та інші параметри, такі як кількість кластерів, максимальна кількість ітерацій та допустима похибка. Результати включають центри кластерів (cntr_train) та матрицю приналежності (u_train), що містить ймовірності приналежності кожного зразка до кожного кластера.

8. Застосування отриманих центрів кластерів (cntr_train) до X_test. Метод fuzz.cluster.cmeans_predict використовує отримані центри кластерів для передбачення приналежності зразків з X_test до кластерів. Результати включають матрицю приналежності (u_test), що містить ймовірності приналежності кожного зразка з X_test до кожного кластера.

Тепер перевіримо розмірність u_train та u_test:

```
u_train shape: (5716073,)
u_test shape: (3, 635120)
```

Рисунок 2.4.2.2 Розмірність u_train та u_test

За допомогою функції predict виведемо масив u_test:

```
(array([[0.02140822, 0.02990875, 0.03060859, ..., 0.02917758, 0.02068882,
        0.53564924],
       [0.06341339, 0.10242862, 0.08711466, ..., 0.08478136, 0.06107368,
        0.23210779],
       [0.91517839, 0.86766263, 0.88227675, ..., 0.88604106, 0.9182375 ,
        0.23224297]]), array([[0.38142722, 0.4541    , 0.39772568, ..., 0.54250066, 0.22213158,
        0.31464057],
       [0.20076836, 0.42265229, 0.01038959, ..., 0.33341433, 0.43322108,
        0.29114175],
       [0.41780441, 0.12324771, 0.59188473, ..., 0.12408502, 0.34464734,
        0.39421768]]), array([[15846849.65113703, 15745146.88351158, 15771253.47072183, ...,
        15919609.4469782 , 15783719.25028777, 29194920.26679619],
       [ 9207521.06296036,  8508151.1389627 ,  9348506.95798217, ...,
         9339139.39661319,  9186501.66720757, 44350924.94092631],
       [ 2423708.12656927,  2923279.94090551,  2937551.29899943, ...,
         2888884.16009826,  2369188.88206488, 44338015.4294705 ]]), array([3.63769933e+19, 9.68629106e+18]), 2, 0.7407809824631
61)
```

Рисунок 2.4.2.3 Масив u_test

U_test містить 3 рядки, матрицю потрібно транспонувати та відкинути третій рядок, бо в сумі три стовпці дадуть стовпець з одиниць. Причина, чому сума цих стовпців дорівнює 1, полягає в особливості результатів нечіткої кластерізації.

У нечіткій кластерізації кожен приклад може належати до кожного кластера з певною ступінню належності. Кожне значення у стовпцях матриці u_test представляє ступінь належності прикладу до певного кластера. Загальна сума ступенів належності для кожного прикладу має бути рівна 1.

Тому виправимо u_test:

```
u_test_transposed shape: (2,)

array([[0.06341336, 0.02140822],
       [0.10242857, 0.02990876],
       [0.08711462, 0.0306086 ],
       ...,
       [0.08478132, 0.02917759],
       [0.06107365, 0.02068882],
       [0.2321078 , 0.53564922]])
```

Рисунок 2.4.2.4 Виправлений u_test

Далі виконаємо об'єднання матриць X_test і u_test_transposed для створення нової матриці X_test_new. Також він об'єднує матриці X_train і u_train для створення нової матриці X_train_new.

У результуючих матрицях X_test_new і X_train_new ми отримуємо розширену версію вхідних даних, де кожний рядок представляє собою поєднання ознак з вихідної матриці і ступеня належності до кожного кластера. За допомогою функції np.hstack() виконується горизонтальне об'єднання матриць, де кожна матриця розміщується вгороджено одна поруч із однією.

В результуючих матрицях X_test_new і X_train_new кількість стовпців збільшилася на два, оскільки були додані стовпці зі ступенями належності до кожного кластера. Це буде потрібно для моделі прогнозування, яка використовує дані кожних кластерів як нову ознаку.

Після цього можемо розробити моделі прогнозування.

Модель на даних нечіткої кластеризації:

```
[95]    validation_0-logloss:0.00077
[96]    validation_0-logloss:0.00077
[97]    validation_0-logloss:0.00077
[98]    validation_0-logloss:0.00077
[99]    validation_0-logloss:0.00077
Accuracy: 0.999770122181635
Logloss: 0.0007661827449868437
[[634322     26]
 [   120    652]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    634348
           1       0.96      0.84      0.90       772

    accuracy                           1.00    635120
   macro avg       0.98      0.92      0.95    635120
weighted avg       1.00      1.00      1.00    635120

F1-score: 0.8993103448275863
AUC-ROC: 0.9222592992630416
```

Рисунок 2.4.2.5 Результат моделі прогнозування на даних нечіткої кластеризації

Результати моделі мають майже такі ж самі результати з моделями К-середніх.

Бачимо більш-менш такий ж самий результат в порівнянні з моделлю методу К-середніх.

Маючи результати моделей на даних нечіткої кластеризації, К-середніх, комбінованого та без кластеризації порівняємо результати загальних моделей:

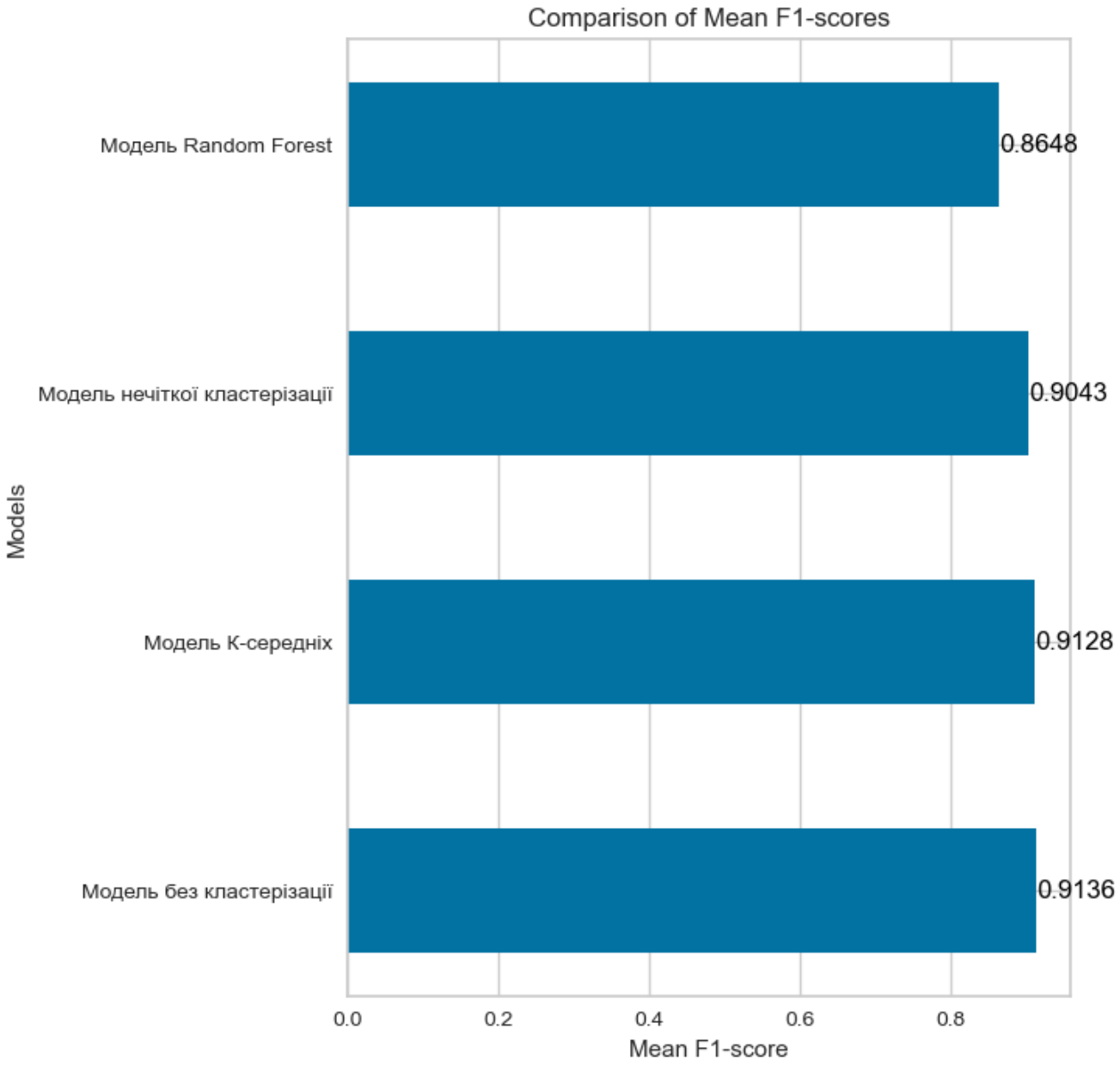

Рисунок 2.4.2.6 Результати загальних моделей прогнозування

Бачимо на графіку, що найкращу якість має модель без кластерізації, випереджаючи модель К-середніх на лише 0.0008%, що є незначною, але явною різницею на графіку між цими моделями. Таким чином ми визначили, що модель без кластеризації позитивно впливає на якість моделі прогнозування XGBoost. Та важливо зазначити, що найкращий потенціал до підвищення якості моделі прогнозування у подальших тестах з даними має модель з комбінованими даними кластерів №0 та №1 через те, що вона має вищі значення F1-оцінки в порівнянні з моделлю без використання кластерів.

ЗАГАЛЬНІ ВИСНОВКИ

В цій дипломній роботі були досліджені та виконані методи класифікації XGBoost та Random Forest, за допомогою кластеризацій K-Means та Fuzzy Clustering. Були проаналізовані та реалізовані чотири спроби підвищення якості моделей XGBoost, але усі вони мають майже однакові або гірші результати якості порівняно з моделлю прогнозування на стандартних тестових та тренувальних даних. Але можемо зазначити, що найкращий потенціал до підвищення якості моделі прогнозування у подальших тестах з даними має модель з комбінованими даними кластерів №0 та №1 через те, що вона має вищі значення F1-оцінки в порівнянні з моделлю без використання кластерів.

В роботі було розглянуто та вивчено методи кодування даних, такі як Label Encoding та One-hot Encoding, що допомогли закодувати категоріальні дані датасету.

В результаті проведеного дослідження виявлено, що комбінація методів кластеризації, прогнозування на даних кластерів з найбільшою кількістю шахрайств та комбінацій даних кластерів з різними параметрами моделі може привести до кращої узгодженості між класами та покращення точності та якості класифікації для незбалансованих наборів даних в задачах Fraud Detection. Також було зазначено значну вищу якість методу XGBoost у порівнянні з методом Random Forest у задачах Fraud Detection з несбалансованими даними та їх великою кількістю.

Загальною метою даної роботи було знайти релевантні методи та алгоритми класифікації незбалансованих наборів даних в задачах Fraud Detection, що дають найкращі результати. Отримані результати та висновки дозволяють рекомендувати використання підходів, які були описані у дипломній роботі, для вирішення задач несбалансованих даних у сфері виявлення шахрайства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning (https://www.researchgate.net/publication/224330873_ADASYN_Adaptive_Synthetic_Sampling_Approach_for_Imbalanced_Learning).

2. Evaluating XGBoost for Balanced and Imbalanced Data: Application to Fraud Detection (https://www.researchgate.net/publication/369556752_Evaluating_XGBoost_for_Balanced_and_Imbalanced_Data_Application_to_Fraud_Detection).

3. https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf.

4. Random Forests (https://www.researchgate.net/profile/Adele-Cutler/publication/236952762_Random_Forests/links/0a85e52f509178323a000000/Random-Forests.pdf?origin=publication_detail).

5. IMBENS: Ensemble Class-imbalanced Learning in Python (https://www.researchgate.net/publication/356602000_IMBENS_Ensemble_Class-imbalanced_Learning_in_Python).

6. Cost-Sensitive Learning vs. Sampling: Which is Best for Handling Unbalanced Classes with Unequal Error Costs? (https://www.researchgate.net/publication/220705031_Cost-Sensitive_Learning_vs_Sampling_Which_is_Best_for_Handling_Unbalanced_Classes_with_Unequal_Error_Costs).

7. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection (https://sci2s.ugr.es/keel/pdf/algorithm/congreso/kubat97addressing.pdf).

8. Learning from Imbalanced Data by He, H., & Garcia, E. A. (2009) (https://www.academia.edu/29164932/Learning_from_Imbalanced_Data).

9. Classification of Imbalanced Data:Review of Methods and Applications (https://www.researchgate.net/publication/350084459_Classification_of_Imbalanced_DataReview_of_Methods_and_Applications).

10. Unstructured Data Analytics: How to Improve Customer Acquisition, Customer Retention, and Fraud Detection and Prevention – by Jean Paul Isson – 26ст.

11. Resampling strategies for imbalanced datasets (https://www.kaggle.com/rafjaa/resamplingstrategies-for-imbalanced-datasets).

12. AFS (Advanced Fraud Detection System)– AEC, 2020. – (https://www.aec.cz/en/afs).

13. The Only Complete Online Banking Fraud Prevention Solution (https://www.threatmark.com).

14. Lutz, Mark. Learning Python. 5 Beijing: O'Reilly, 2013.

15. "Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python" by Manohar Swamynathan.

16. "Python Data Science Handbook: Essential Tools for Working with Data" by Jake VanderPlas

ДОДАТКИ

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кількість аркушів	Примітки
1				
2		Документація		
3				
4	САУ.КР.YY.ZZ.ПЗ	Пояснювальна записка	N1	Формат А4
5				
6		Демонстраційний матеріал	N2	Презентація на CD-R
7				
8		Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.YY.ZZ.ДА.ПЗ.			
Змін.	Аркуш	№ докум.	Підпис	Дата				
Розроб.		ПІБ			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Аркушів
К. розд.		ПІБ						
Керівн.		ПІБ				НТУ «ДП», 12; 124-18-1		
Н.контр.		ПІБ						
Зав. каф.		ПІБ						

Запис **САУ.КР.YY.ZZ.ПЗ** означає наступне:
САУ – код випускаючої кафедри;
КР – кваліфікаційна робота;
N1 – загальна кількість сторінок пояснювальної записки кваліфікаційної роботи з додатками;
N2 – кількість аркушів демонстраційного матеріалу (слайдів презентації);
YY – рік захисту кваліфікаційної роботи в ЕК (наприклад “22”);
ZZ – номер теми студента в наказі про затвердження теми кваліфікаційної роботи (наприклад “06”);
ПЗ – пояснювальна записка;
ДА – додаток А;
12 – код галузі «Інформаційні технології».

ДОДАТОК Б

Відгук
на кваліфікаційну роботу бакалавра
студента групи *124 – 19 – 2*
спеціальності *124 Системний аналіз*

Тема кваліфікаційної роботи: _____
_____
Обсяг кваліфікаційної роботи _____ стор.
Мета кваліфікаційної роботи: _____
_____
Актуальність теми _____
_____
_____
Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки _____
_____
_____
Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра. Оригінальність наукових рішень полягає в _____
_____
_____
_____
Практичне значення результатів кваліфікаційної роботи полягає в _____
_____
_____
Висновки підтверджують можливість використання результатів роботи в _____
_____
_____
Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі (*в разі невідповідності – вказати*)
У роботі відзначено такі недоліки: _____
_____
_____
Кваліфікаційна робота в цілому заслуговує оцінки:_____

З урахуванням висловлених зауважень автор (не) заслуговує присвоєння освітньої кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи бакалавра,
науковий ступінь, вчене звання, посада _____ / ПІБ

ДОДАТОК В

```
import pandas as pd #To work with dataset
import numpy as np #Math library
import matplotlib.gridspec as gridspec
import seaborn as sns #Graph library that use matplot in background
import matplotlib.pyplot as plt #to plot some parameters in seaborn
import warnings
import plotly.graph_objects as go

from pandas import Series
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer,IterativeImputer
from sklearn.preprocessing import RobustScaler
import xgboost as xgb

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline, Pipeline,FeatureUnion
from sklearn.manifold import TSNE
from sklearn.metrics import roc_auc_score, average_precision_score
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve,confusion_matrix
from datetime import datetime, date
from sklearn.linear_model import ElasticNet, Lasso, BayesianRidge,
LassoLarsIC
from sklearn.linear_model import LinearRegression, RidgeCV
```

```
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
import smogn
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.kernel_ridge import KernelRidge
from sklearn.ensemble import GradientBoostingRegressor,RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
# For training random forest model
import lightgbm as lgb
from scipy import sparse
from sklearn.neighbors import KNeighborsRegressor
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
# Model selection
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression , f_classif, chi2
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import mutual_info_classif,VarianceThreshold

from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

from scipy import stats, optimize, interpolate

from lightgbm import LGBMClassifier
import lightgbm as lgbm
from catboost import CatBoostRegressor, CatBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from xgboost import XGBClassifier
from sklearn import set_config
from itertools import combinations
# Cluster :
from sklearn.cluster import MiniBatchKMeans
from yellowbrick.cluster import KElbowVisualizer
#import smong
import category_encoders as ce
import warnings
import optuna
from joblib import Parallel, delayed
import joblib
from sklearn import set_config
set_config(display='diagram')
warnings.filterwarnings('ignore')
import klib
class FeaturesEngineer(BaseEstimator, TransformerMixin):
```

```
        def fit(self, X, y=None):
            return self
        def mc_cat(self,val):
            if val == 0:
                return 0
            elif val == 1 or val == 14:
                return 1
            else:
                return 2
        def transform(self, X, y=None):
            # Calculate some metrics across rows
            X["num_missing"]  = X.isnull().sum(axis=1)
            X["std_row"] = X.std(axis=1)
            X["sem_row"] = X.sem(axis=1)
            X["abs_sum_row"] = X.abs().sum(axis=1)
            X["mean_row"] = X.mean(axis=1)
            X["max_row"]= X.max(axis=1)
            X["min_row"]= X.min(axis=1)
            X['mc_cat'] = X['num_missing'].apply(self.mc_cat)
            return X


        from sklearn.cluster import MiniBatchKMeans, KMeans
    class MiniKmeansTransformer(BaseEstimator, TransformerMixin):
        def __init__(self, num_clusters = 6):
            self.num_clusters = num_clusters
            if self.num_clusters > 0:
                self.kmeans = MiniBatchKMeans(n_clusters=self.num_clusters,
random_state=0)
```

```
        def fit(self, X, y=None):
            if self.num_clusters > 0:
                self.kmeans.fit(X)
            return self

        def transform(self, X, y=None):
            pred_classes =self.kmeans.predict(X).reshape(-1,1)
            pred_classes=(pred_classes - np.min(pred_classes)) / (np.max(pred_classes) - np.min(pred_classes))
            return np.hstack((X, pred_classes))


    class ColumnSelector(BaseEstimator, TransformerMixin):
        """Select only specified columns."""
        def __init__(self, positions):
            self.positions = positions
        def fit(self, X, y=None):
            return self
        def transform(self, X):
            #return np.array(X)[:, self.positions]
            return X.loc[:, self.positions]

    # Quantile Outlier Handling
    class OutlierReplace(BaseEstimator,TransformerMixin):
        def __init__(self,factor=1.5):
            self.factor = factor

        def outlier_removal(self,X,y=None):
            X = pd.Series(X).copy()
```

```
        qmin=X.quantile(0.05)
        qmax=X.quantile(0.95)
        q1 = X.quantile(0.25)
        q3 = X.quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - (self.factor * iqr)
        upper_bound = q3 + (self.factor * iqr)
        #X.loc[((X < lower_bound) | (X > upper_bound))] = np.nan
        X.loc[X < lower_bound] = qmin
        X.loc[X > upper_bound] = qmax
        return pd.Series(X)

    def fit(self,X,y=None):
        return self

    def transform(self,X,y=None):
        return X.apply(self.outlier_removal)
train = pd.read_csv("transactions_train.csv")
# Preview the data
train.head(3)
# Author : https://www.kaggle.com/gemartin/load-data-reduce-memory-usage
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))
```

```
        for col in df.columns:
            col_type = df[col].dtype
            name =df[col].dtype.name

            if col_type != object and col_type.name != 'category':
            #if name != "category":
                c_min = df[col].min()
                c_max = df[col].max()
                if str(col_type)[:3] == 'int':
                    if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)
                else:
                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                        df[col] = df[col].astype(np.float16)
                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                        df[col] = df[col].astype(np.float32)
                    else:
                        df[col] = df[col].astype(np.float64)
            else:
```

```
                df[col] = df[col].astype('category')


            end_mem = df.memory_usage().sum() / 1024**2
            print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
            print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))


            return df
        # train= reduce_mem_usage(train)
        train.info()
        train.columns
        train.isnull().sum().values
        # summarize the number of rows with missing values for each column
        for i in range(train.shape[1]):
            # count number of rows with missing values
            n_miss = train.iloc[:,i].isnull().sum()
            perc = n_miss / train.shape[0] * 100
            print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
        klib.missingval_plot(train)
        train.duplicated().sum()
        len(train)-len(train.drop_duplicates())
        skew =train.skew().sort_values(ascending =False )
        skew_df= pd.DataFrame({'skew':skew})
        skew_df.head(10)
        skew_df[(skew_df['skew']>=1) |(skew_df['skew']<=-1) ].index
        ax = sns.distplot(train['amount'])
        sns.boxplot(data=train['amount'], saturation=.3)
        amount_corrected= np.log(train['amount' ]+1)
```

```
print(train['amount'].skew())


print(amount_corrected.skew())
ax = sns.distplot(amount_corrected)
kurtosis= pd.DataFrame(train.kurtosis(),columns=['Kurtosis'])
kurtosis.head(8)
kurtosis[(kurtosis['Kurtosis']>=3) |(kurtosis['Kurtosis']<=-3) ].index
amount_corrected.kurtosis()
var= train.var().sort_values(ascending =True )
var_df= pd.DataFrame({'var':var})
var_df.head(10)
Q1 = train.quantile(0.25)
Q3 = train.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
train_num=train.select_dtypes(exclude=['category'])
df_out = train_num[~((train_num < (Q1 - 1.5 * IQR))|(train_num > (Q3 + 1.5 *
IQR))).any(axis=1)]
print(df_out.shape)
del train_num
del df_out
train.shape
train.describe().T
sns.pairplot(train.iloc[0:50000], hue= 'isFraud')
plot = klib.corr_plot(train,  figsize=(12,10))
klib.corr_plot(train, split='pos')
klib.dist_plot(train[['step',  'amount', 'nameOrig', 'oldbalanceOrig',
      'newbalanceOrig', 'nameDest', 'oldbalanceDest', 'newbalanceDest']])
g= sns.countplot(x='type', data=train)
```

```
plt.xticks(rotation=45)

train[train.select_dtypes(['int64','int16','float32','float64','int8']).columns] = train[train.select_dtypes(['int64','int16','float32','float64','int8']).columns].apply(pd.to_numeric)

train[train.select_dtypes(['object','category']).columns] = train.select_dtypes(['object','category']).apply(lambda x: x.astype('category'))

target= "isFraud"

X = train.drop(target, axis='columns')# axis=1

y = train[target].to_numpy()

cat_columns = X.select_dtypes(exclude=['int64','int16','float32','float64','int8']).columns

num_columns = X.select_dtypes(include=['int64','int16','float32','float64','int8']).columns

f, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 12))

f.suptitle('amount', fontsize=16)

g = sns.kdeplot(train['amount'], shade=True, label="%.2f"%(train['amount'].skew()), ax=axes[0])

g = g.legend(loc="best")

stats.probplot(train['amount'], plot=axes[1])

sns.boxplot(x='amount', data=train, orient='h', ax=axes[2]);

plt.tight_layout()

plt.show()

i = 1

plt.figure()

fig, ax = plt.subplots(3, 2,figsize=(20, 24))

for feature in num_columns:

    plt.subplot(3, 2,i)

    sns.histplot(train[feature],color="red", kde=True,bins=100, label='train')

    # sns.histplot(test[feature],color="olive", kde=True,bins=100, label='test')

    plt.xlabel(feature, fontsize=9); plt.legend()
```

```
    i += 1
plt.show()
train.corr().style.background_gradient(cmap='viridis')
train.corr()['isFraud'][:-1].plot.barh(figsize=(8,6),alpha=.6,color='darkblue')
plt.xlim(-.075,.075);
plt.xticks([-0.065, -0.05 , -0.025,  0.   ,  0.025,  0.05 ,  0.065],
           [str(100*i)+'%' for i in [-0.065, -0.05 , -0.025,  0.   ,  0.025,  0.05 , 0.065]],fontsize=12)
plt.title('Correlation between target and numerical variables',fontsize=14);
v0 = sns.color_palette(palette='viridis').as_hex()[0]
fig = plt.figure(figsize=(18,6))
sns.boxplot(data=train[num_columns], color=v0,saturation=.5);
plt.xticks(fontsize= 14)
plt.title('Box plot of train numerical columns', fontsize=16);
fig = plt.figure(figsize=(10,5))
sns.barplot(y=train[cat_columns].nunique().values, x=train[cat_columns].nunique().index, color='blue', alpha=.5)
plt.xticks(rotation=0)
plt.title('Number of categorical unique values',fontsize=16);
labels = train['type'].astype('category').cat.categories.tolist()
counts = train['type'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the % on plot
ax1.axis('equal')
plt.show()
i = 1
plt.figure()
```

```
fig, ax = plt.subplots(1, 1,figsize=(10,10))
for feature in ['type']:
    plt.subplot(1, 1,i)
    sns.histplot(train[feature],color="blue", label='train')
    #sns.histplot(test[feature],color="olive", label='test')
    plt.xlabel(feature, fontsize=9); plt.legend()
    i += 1
plt.show()
train.isFraud.value_counts()
plt.figure()
sns.countplot(train['isFraud'], label='IsFraud')
plt.xlabel(feature, fontsize=9); plt.legend()
plt.xticks(rotation=45)
plt.show()
fig = plt.figure(figsize=(30,10))
grid =  gridspec.GridSpec(2,3,figure=fig,hspace=.4,wspace=.4)
n =0
for i in range(2):
    for j in range(3):
        ax = fig.add_subplot(grid[i, j])
        sns.violinplot(data =  train.iloc[0:50000], y = num_columns[n] , x ='isFraud'
,ax=ax, alpha =.7, fill=True,palette='viridis')
        ax.set_title(num_columns[n],fontsize=14)
        ax.set_xlabel('')
        ax.set_ylabel('')
        n += 1

fig.suptitle('Violin plot of target with numerical features', fontsize=16,y=.93);
fig = plt.figure(figsize=(26,10))
```

```
grid =  gridspec.GridSpec(3,2,figure=fig,hspace=.2,wspace=.2)
n =0
for i in range(3):
   for j in range(2):
      ax = fig.add_subplot(grid[i, j])
      sns.kdeplot(data = train.iloc[0:800000,:], y = num_columns[n], hue = 'isFraud',ax=ax, alpha =.7, fill=False)
      ax.set_title(num_columns[n],fontsize=14)
      ax.set_xlabel('')
      ax.set_ylabel('')
      n += 1

fig.suptitle('KDE plot of train target with  features', fontsize=16,y=.93);
%%time
m = TSNE()
df_numeric =train.iloc[0:8000]._get_numeric_data()
df_numeric=df_numeric.dropna()
X_train =RobustScaler().fit_transform(df_numeric)
del df_numeric
# Fit and transform the t-SNE model on the numeric dataset
tsne_features = m.fit_transform(X_train)
print(tsne_features.shape)
trainessai=train.iloc[0:8000]
trainessai['x']=tsne_features[:, 0]
trainessai['y']=tsne_features[:, 1]
# Color the points according to Army Component
sns.scatterplot(x='x', y='y', hue='isFraud', data=trainessai)
# Show the plot
plt.show
```

```
# Split the dataset and labels into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,random_state=0,stratify=y )
print("{} rows in test set vs. {} in training set. {} Features.".format(X_test.shape[0], X_train.shape[0], X_test.shape[1]))
print(y_train.shape)
```

ДОДАТОК Г

```
import pandas as pd #To work with dataset
import numpy as np #Math library
import matplotlib.gridspec as gridspec
import seaborn as sns #Graph library that use matplot in background
import matplotlib.pyplot as plt #to plot some parameters in seaborn
import warnings
import plotly.graph_objects as go

from pandas import Series
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer,IterativeImputer
from sklearn.preprocessing import RobustScaler
import xgboost as xgb

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline, Pipeline,FeatureUnion
from sklearn.manifold import TSNE
from sklearn.metrics import roc_auc_score, average_precision_score
```

```
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve,confusion_matrix
from datetime import datetime, date
from sklearn.linear_model import ElasticNet, Lasso, BayesianRidge, LassoLarsIC
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
import smogn
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.kernel_ridge import KernelRidge
from sklearn.ensemble import GradientBoostingRegressor,RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
# For training random forest model
import lightgbm as lgb
from scipy import sparse
from sklearn.neighbors import KNeighborsRegressor
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
# Model selection
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression , f_classif, chi2
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import mutual_info_classif,VarianceThreshold

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

from scipy import stats, optimize, interpolate

from lightgbm import LGBMClassifier
import lightgbm as lgbm
from catboost import CatBoostRegressor, CatBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from xgboost import XGBClassifier
from sklearn import set_config
from itertools import combinations
# Cluster :
from sklearn.cluster import MiniBatchKMeans
from yellowbrick.cluster import KElbowVisualizer
#import smong
import category_encoders as ce
import warnings
import optuna
```

```
from joblib import Parallel, delayed
import joblib
from sklearn import set_config
set_config(display='diagram')
warnings.filterwarnings('ignore')
import klib
class FeaturesEngineer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def mc_cat(self,val):
        if val == 0:
            return 0
        elif val == 1 or val == 14:
            return 1
        else:
            return 2
    def transform(self, X, y=None):
        # Calculate some metrics across rows
        X["num_missing"]  = X.isnull().sum(axis=1)
        X["std_row"] = X.std(axis=1)
        X["sem_row"] = X.sem(axis=1)
        X["abs_sum_row"] = X.abs().sum(axis=1)
        X["mean_row"] = X.mean(axis=1)
        X["max_row"]= X.max(axis=1)
        X["min_row"]= X.min(axis=1)
        X['mc_cat'] = X['num_missing'].apply(self.mc_cat)
        return X
```

```
        from sklearn.cluster import MiniBatchKMeans, KMeans
    class MiniKmeansTransformer(BaseEstimator, TransformerMixin):
        def __init__(self, num_clusters = 6):
            self.num_clusters = num_clusters
            if self.num_clusters > 0:
                self.kmeans = MiniBatchKMeans(n_clusters=self.num_clusters, random_state=0)

        def fit(self, X, y=None):
            if self.num_clusters > 0:
                self.kmeans.fit(X)
            return self

        def transform(self, X, y=None):
            pred_classes =self.kmeans.predict(X).reshape(-1,1)
            pred_classes=(pred_classes - np.min(pred_classes)) / (np.max(pred_classes) - np.min(pred_classes))
            return np.hstack((X, pred_classes))


    class ColumnSelector(BaseEstimator, TransformerMixin):
        """Select only specified columns."""
        def __init__(self, positions):
            self.positions = positions
        def fit(self, X, y=None):
            return self
        def transform(self, X):
            #return np.array(X)[:, self.positions]
            return X.loc[:, self.positions]
```

```
# Quantile Outlier Handling
class OutlierReplace(BaseEstimator,TransformerMixin):
    def __init__(self,factor=1.5):
        self.factor = factor

    def outlier_removal(self,X,y=None):
        X = pd.Series(X).copy()
        qmin=X.quantile(0.05)
        qmax=X.quantile(0.95)
        q1 = X.quantile(0.25)
        q3 = X.quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - (self.factor * iqr)
        upper_bound = q3 + (self.factor * iqr)
        #X.loc[((X < lower_bound) | (X > upper_bound))] = np.nan
        X.loc[X < lower_bound] = qmin
        X.loc[X > upper_bound] = qmax
        return pd.Series(X)

    def fit(self,X,y=None):
        return self

    def transform(self,X,y=None):
        return X.apply(self.outlier_removal)
# Author : https://www.kaggle.com/gemartin/load-data-reduce-memory-usage
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
```

```
    to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype
        name =df[col].dtype.name

        if col_type != object and col_type.name != 'category':
        #if name != "category":
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
```

```
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
            else:
                df[col] = df[col].astype('category')

        end_mem = df.memory_usage().sum() / 1024**2
        print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
        print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

        return df
    # train= reduce_mem_usage(train)
    train = pd.read_csv("transactions_train.csv")
    # Preview the data
    train.head(3)
    train.info()
    train.columns
    train.isnull().sum().values
    # summarize the number of rows with missing values for each column
    for i in range(train.shape[1]):
        # count number of rows with missing values
        n_miss = train.iloc[:,i].isnull().sum()
        perc = n_miss / train.shape[0] * 100
        print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
    train.duplicated().sum()
    len(train)-len(train.drop_duplicates())
```

```
target= "isFraud"
X = train.drop(target, axis='columns')# axis=1
y = train[target].to_numpy()
cat_columns= X.select_dtypes(exclude=['int64','int16','float32','float64','int8']).columns
num_columns= X.select_dtypes(include=['int64','int16','float32','float64','int8']).columns
train.isFraud.value_counts()
# Split the dataset and labels into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,random_state=0,stratify=y )
print("{} rows in test set vs. {} in training set. {} Features.".format(X_test.shape[0], X_train.shape[0], X_test.shape[1]))
print(y_train.shape)
import xgboost as xgb
from xgboost import plot_importance
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X_train['type'] = le.fit_transform(X_train['type'])
X_train['nameOrig'] = le.fit_transform(X_train['nameOrig'])
X_train['nameDest'] = le.fit_transform(X_train['nameDest'])
X_test['type'] = le.fit_transform(X_test['type'])
X_test['nameOrig'] = le.fit_transform(X_test['nameOrig'])
X_test['nameDest'] = le.fit_transform(X_test['nameDest'])
print(X_train.dtypes)
print(X_test.dtypes)
from xgboost import XGBClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import classification_report, f1_score
```

```
# create XGBoost classifier object
xgb_model_no_clust = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model_no_clust.fit(X_train, y_train, eval_set=[(X_test, y_test)],
            early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred_no_clust = xgb_model_no_clust.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred_no_clust)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test, xgb_model_no_clust.predict_proba(X_test))
print("Logloss:", logloss)
print(confusion_matrix(y_test, y_pred_no_clust))
print(classification_report(y_test, y_pred_no_clust))
# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test, y_pred_no_clust))
print("AUC-ROC:", roc_auc_score(y_test, y_pred_no_clust))
from sklearn.preprocessing import StandardScaler

# Создание экземпляра класса StandardScaler
scaler = StandardScaler()

# Нормализация признаков в обучающем наборе данных
```

```
X_train_scaled = scaler.fit_transform(X_train)

# Применение нормализации к признакам в тестовом наборе данных
X_test_scaled = scaler.transform(X_test)
from sklearn.cluster import MiniBatchKMeans
import matplotlib.pyplot as plt

# List of inertias
inertias = []
# Range of k values to test
k_range = range(1, 10)
for k in k_range:
    # Create a MiniBatchKMeans instance with k clusters
    model = MiniBatchKMeans(n_clusters=k, batch_size=1000, max_iter=100)
    # Fit the model to the data
    model.fit(X_train_scaled)
    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

# Plot the elbow curve
plt.plot(k_range, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('The Elbow Method showing the optimal k')
plt.show()
# Calculate the differences between each consecutive inertia value
diff = np.diff(inertias)
```

```
# Calculate the second differences between each consecutive inertia value
diff2 = np.diff(diff)

# Find the index of the elbow point, which is the point of maximum curvature
elbow_index = np.argmax(diff2) + 2

# Print the best number of clusters
print(f"The best number of clusters is {elbow_index}")
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import LabelEncoder

# Create an instance of LabelEncoder
le = LabelEncoder()

# Fit and transform the categorical labels in y_train
y_train_encoded = le.fit_transform(y_train)

# Transform the categorical labels in y_test
y_test_encoded = le.transform(y_test)

# Instantiate the k-means clustering algorithm
kmeans = MiniBatchKMeans(n_clusters=elbow_index, batch_size=1000, max_iter=100)

# Fit the algorithm to the training data
kmeans.fit(X_train_scaled)

# Obtain cluster assignments for each data point in the training set
cluster_labels_train = kmeans.predict(X_train_scaled)
```

```
# Obtain cluster assignments for each data point in the test set
cluster_labels_test = kmeans.predict(X_test_scaled)

# Create a new training set that includes only data points belonging to cluster 0
X_train_cluster0 = X_train_scaled[cluster_labels_train == 0]
y_train_cluster0 = y_train_encoded[cluster_labels_train == 0]
# Create a new test set that includes only data points belonging to cluster 0
X_test_cluster0 = X_test_scaled[cluster_labels_test == 0]
y_test_cluster0 = y_test_encoded[cluster_labels_test == 0]

X_train_cluster1 = X_train_scaled[cluster_labels_train == 1]
y_train_cluster1 = y_train_encoded[cluster_labels_train == 1]

X_test_cluster1 = X_test_scaled[cluster_labels_test == 1]
y_test_cluster1 = y_test_encoded[cluster_labels_test == 1]

X_train_cluster2 = X_train_scaled[cluster_labels_train == 2]
y_train_cluster2 = y_train_encoded[cluster_labels_train == 2]

X_test_cluster2 = X_test_scaled[cluster_labels_test == 2]
y_test_cluster2 = y_test_encoded[cluster_labels_test == 2]
# Calculate the fraud percentage for Cluster 0
fraud_percentage_cluster0 = (np.sum(y_test_cluster0) / np.sum(y_test_encoded))
* 100

# Calculate the fraud percentage for Cluster 1
fraud_percentage_cluster1 = (np.sum(y_test_cluster1) / np.sum(y_test_encoded))
* 100
```

```
# Calculate the fraud percentage for Cluster 2
fraud_percentage_cluster2 = (np.sum(y_test_cluster2) / np.sum(y_test_encoded)) * 100



# Print the fraud percentages for each cluster
print("Fraud percentage in Cluster 0:", fraud_percentage_cluster0)
print("Fraud percentage in Cluster 1:", fraud_percentage_cluster1)
print("Fraud percentage in Cluster 2:", fraud_percentage_cluster2)

sum_per = fraud_percentage_cluster0 + fraud_percentage_cluster1 + fraud_percentage_cluster2
print(sum_per)
relative_size_cluster0 = len(X_test_cluster0) / len(X_test)
relative_size_cluster1 = len(X_test_cluster1) / len(X_test)
relative_size_cluster2 = len(X_test_cluster2) / len(X_test)

relative_fraud_percentage_cluster0 = (np.sum(y_test_cluster0) / len(y_test_cluster0)) * 100
relative_fraud_percentage_cluster1 = (np.sum(y_test_cluster1) / len(y_test_cluster1)) * 100
relative_fraud_percentage_cluster2 = (np.sum(y_test_cluster2) / len(y_test_cluster2)) * 100

print("Relative size of Cluster 0:", relative_size_cluster0)
print("Relative size of Cluster 1:", relative_size_cluster1)
print("Relative size of Cluster 2:", relative_size_cluster2)
print("Relative fraud perc in Cluster 0:", relative_fraud_percentage_cluster0)
print("Relative fraud perc in Cluster 1:", relative_fraud_percentage_cluster1)
```

```
print("Relative fraud perc in Cluster 2:", relative_fraud_percentage_cluster2)

X_train_combined_02 = np.concatenate((X_train_cluster0, X_train_cluster2), axis=0)

y_train_combined_02 = np.concatenate((y_train_cluster0, y_train_cluster2), axis=0)


X_test_combined_02 = np.concatenate((X_test_cluster0, X_test_cluster2), axis=0)

y_test_combined_02 = np.concatenate((y_test_cluster0, y_test_cluster2), axis=0)


X_train_combined_01 = np.concatenate((X_train_cluster0, X_train_cluster1), axis=0)

y_train_combined_01 = np.concatenate((y_train_cluster0, y_train_cluster1), axis=0)


X_test_combined_01 = np.concatenate((X_test_cluster0, X_test_cluster1), axis=0)

y_test_combined_01 = np.concatenate((y_test_cluster0, y_test_cluster1), axis=0)

train_df = pd.DataFrame({'Cluster': cluster_labels_train, 'isFraud': y_train})

# Додавання стовпця "cluster" зі значеннями

train_df['cluster'] = cluster_labels_train

train_df['cluster'] = cluster_labels_train.astype('object')

red = sns.light_palette("red", as_cmap=True)


cross_tab=pd.crosstab(train_df['cluster'], train_df['isFraud'], margins = True)


H=cross_tab/cross_tab.loc["All"] # Divide by column totals


H.style.background_gradient(cmap=red)

# Create a DataFrame containing the cluster labels and the corresponding fraud labels
```

```
fraud_df = pd.DataFrame({'Cluster': cluster_labels_train, 'isFraud': y_train})

# Count the number of fraudulent transactions in each cluster
fraud_counts = fraud_df.groupby('Cluster')['isFraud'].sum()

# Print the number of fraudulent transactions in each cluster
print(fraud_counts)
import matplotlib.pyplot as plt

# Count the number of data points in each cluster
cluster_counts = pd.Series(cluster_labels_train).value_counts()

# Create a bar plot to visualize the cluster counts
plt.bar(cluster_counts.index, cluster_counts.values)
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.title('Number of Data Points in Each Cluster')

# Add count values as text annotations
for i, count in enumerate(cluster_counts.values):
    plt.text(cluster_counts.index[i], count, str(count), ha='center', va='bottom')

plt.show()
fraud_counts = fraud_df.groupby('Cluster')['isFraud'].sum()

plt.bar(fraud_counts.index, fraud_counts.values)
plt.xlabel('Cluster')
plt.ylabel('Count of Fraudulent Transactions')
```

```
plt.title('Number of Fraudulent Transactions in Each Cluster')

# Додати числа як текстові анотації
for i, count in enumerate(fraud_counts.values):
    plt.annotate(str(count), xy=(fraud_counts.index[i], count), ha='center', va='bottom')

plt.show()
from xgboost import XGBClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import classification_report, f1_score
# Создание экземпляра класса XGBClassifier
xgb_model_clustered = XGBClassifier(objective='binary:logistic')

# Создание новых признаков с использованием меток кластеров
X_train_clustered = np.hstack((X_train_scaled, cluster_labels_train.reshape(-1, 1)))
X_test_clustered = np.hstack((X_test_scaled, cluster_labels_test.reshape(-1, 1)))

# Обучение модели на обучающих данных с учетом кластерных признаков
xgb_model_clustered.fit(X_train_clustered, y_train_encoded, eval_set=[(X_test_clustered, y_test_encoded)],
                        early_stopping_rounds=10, verbose=1)

# Прогнозирование меток на тестовых данных с учетом кластерных признаков
y_pred_clustered = xgb_model_clustered.predict(X_test_clustered)

# Оценка качества модели на тестовых данных
accuracy = accuracy_score(y_test_encoded, y_pred_clustered)
```

```
print("Accuracy:", accuracy)

# Построение отчета по классификации
classification_report = classification_report(y_test_encoded, y_pred_clustered)
print(confusion_matrix(y_test_encoded, y_pred_clustered))
print(classification_report)

# Оценка F1-меры
f1score = f1_score(y_test_encoded, y_pred_clustered)
print("F1-score:", f1score)

# Оценка AUC-ROC
auc_roc = roc_auc_score(y_test_encoded, y_pred_clustered)
print("AUC-ROC:", auc_roc)
# Create XGBoost classifier object with reduced depth and number of trees
xgb_model_stand_01 = XGBClassifier(objective='binary:logistic')

# Fit the model on training data with validation set specified
xgb_model_stand_01.fit(X_train_combined_01, y_train_combined_01,
eval_set=[(X_test_combined_01, y_test_combined_01)],
        early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred_combined_01_stand =
xgb_model_stand_01.predict(X_test_combined_01)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_combined_01, y_pred_combined_01_stand)
print("Accuracy:", accuracy)
```

```
# Print the logloss
logloss = log_loss(y_test_combined_01, xgb_model_stand_01.predict_proba(X_test_combined_01))
print("Logloss:", logloss)
print(confusion_matrix(y_test_combined_01, y_pred_combined_01_stand))
print(classification_report(y_test_combined_01, y_pred_combined_01_stand))
# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_combined_01, y_pred_combined_01_stand))
print("AUC-ROC:", roc_auc_score(y_test_combined_01, y_pred_combined_01_stand))
# Create XGBoost classifier object with reduced depth and number of trees
xgb_model_01 = XGBClassifier(objective='binary:logistic', max_depth=5, n_estimators=500, learning_rate = 0.1, subsample =1, colsample_bytree = 1, tree_method = 'auto')

# Fit the model on training data with validation set specified
xgb_model_01.fit(X_train_combined_01, y_train_combined_01, eval_set=[(X_test_combined_01, y_test_combined_01)],
                 early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred_combined_01 = xgb_model_01.predict(X_test_combined_01)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_combined_01, y_pred_combined_01)
print("Accuracy:", accuracy)

# Print the logloss
```

```
    logloss = log_loss(y_test_combined_01, xgb_model_01.predict_proba(X_test_combined_01))
    print("Logloss:", logloss)
    print(confusion_matrix(y_test_combined_01, y_pred_combined_01))
    print(classification_report(y_test_combined_01, y_pred_combined_01))
    # Оценка качества модели на тестовых данных
    print("F1-score:", f1_score(y_test_combined_01, y_pred_combined_01))
    print("AUC-ROC:", roc_auc_score(y_test_combined_01, y_pred_combined_01))
    from sklearn.ensemble import RandomForestClassifier

    # create Random Forest classifier object
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

    # fit the model on training data
    rf_model.fit(X_train_scaled, y_train_encoded)

    # Make predictions on the clustered test data
    y_pred_rf = rf_model.predict(X_test_scaled)

    # Evaluate the model's accuracy
    accuracy = accuracy_score(y_test_encoded, y_pred_rf)
    print("Accuracy:", accuracy)

    # Print the logloss
    logloss = log_loss(y_test_encoded, rf_model.predict_proba(X_test_scaled))
    print("Logloss:", logloss)

    # Print the confusion matrix and classification report
    print(confusion_matrix(y_test_encoded, y_pred_rf))
```

```
print(classification_report(y_test_encoded, y_pred_rf))


# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_encoded, y_pred_rf))
print("AUC-ROC:", roc_auc_score(y_test_encoded, y_pred_rf))
# perform 5-fold cross validation and print average f1-score
f1_scores_no_clust = cross_val_score(xgb_model_no_clust, X_train, y_train, cv=5, scoring='f1')
print("F1-scores:", f1_scores_no_clust)
print("Average no clust F1-score:", f1_scores_no_clust.mean())
# perform 5-fold cross validation and print average f1-score
f1_scores_clustered = cross_val_score(xgb_model_clustered, X_train_clustered, y_train_encoded, cv=5, scoring='f1')
print("F1-scores:", f1_scores_clustered)
print("Average clustered F1-score:", f1_scores_clustered.mean())
# perform 5-fold cross validation and print average f1-score
f1_scores_0 = cross_val_score(xgb_model_clust_0, X_train_cluster0, y_train_cluster0, cv=5, scoring='f1')
print("F1-scores:", f1_scores_0)
print("Average cluster 0 F1-score:", f1_scores_0.mean())
# perform 5-fold cross validation and print average f1-score
f1_scores_combined = cross_val_score(xgb_model_01, X_train_combined_01, y_train_combined_01, cv=5, scoring='f1')
print("F1-scores:", f1_scores_combined)
print("Average combined 02 F1-score:", f1_scores_combined.mean())
from sklearn.model_selection import cross_val_score


f1_scores_rf = cross_val_score(rf_model, X_train_scaled, y_train_encoded, cv=5, scoring='f1')
print("F1-scores:", f1_scores_rf)
```

```
print("Average F1-score:", f1_scores_rf.mean())
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6,8)
xgb.plot_importance(xgb_model_no_clust)
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6,8)
xgb.plot_importance(xgb_model_clust_0)
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6,8)
xgb.plot_importance(xgb_model_clustered)
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6,8)
xgb.plot_importance(xgb_model_01)
import numpy as np
import matplotlib.pyplot as plt


# Compute the mean of F1-scores
mean_k_means_results = 0.9128428823470383
mean_no_clust_results = 0.9136234971354481
mean_c0_results = 0.8357740338706107
mean_rf_results = 0.8647661869854654
mean_k_means_combined_results = 0.911019488604009
# Model names
model_names = ['Модель К-середніх', 'Модель кластера №0', 'Модель Random Forest', 'Модель Комбінована', 'Модель без кластерізації']


# Create the horizontal bar plot with thinner bars
plt.barh(model_names, [mean_k_means_results, mean_c0_results, mean_rf_results,mean_k_means_combined_results,mean_no_clust_results], height=0.5)
```

```
plt.xlabel('Mean F1-score')
plt.ylabel('Models')
plt.title('Comparison of Mean F1-scores')

# Display exact values on the bars
for i, v in enumerate([mean_k_means_results, mean_c0_results, mean_rf_results,mean_k_means_combined_results,mean_no_clust_results]):
    plt.text(v, i, str(round(v, 4)), color='black', va='center')

plt.show()
import numpy as np
import matplotlib.pyplot as plt

# Compute the mean of F1-scores
mean_k_means_results = 0.9128428823470383
mean_rf_results = 0.8647661869854654
mean_Fuzzy_results = 0.9043113549275323
mean_no_clust_results = 0.9136234971354481
mean_k_means_combined_results = 0.911019488604009

# Model names
model_names = ['Модель без кластерізації', 'Модель Random Forest', 'Модель нечіткої кластерізації', 'Модель К-середніх','Модель Комбінована']

# Mean F1-scores
mean_scores = [mean_no_clust_results, mean_rf_results, mean_Fuzzy_results, mean_k_means_results]

# Sort the model names and mean scores in descending order
```

```
sorted_indices = np.argsort(mean_scores)[::-1]  # Indices that would sort the scores in descending order
sorted_model_names = [model_names[i] for i in sorted_indices]
sorted_mean_scores = [mean_scores[i] for i in sorted_indices]

# Create the horizontal bar plot with sorted values
plt.barh(sorted_model_names, sorted_mean_scores, height=0.5)
plt.xlabel('Mean F1-score')
plt.ylabel('Models')
plt.title('Comparison of Mean F1-scores')

# Display exact values on the bars
for i, v in enumerate(sorted_mean_scores):
    plt.text(v, i, str(round(v, 4)), color='black', va='center')

plt.show()
import matplotlib.pyplot as plt

# Define the f1_scores and labels
f1_scores = [f1_scores_no_clust, f1_scores_combined]
labels = ['No Clustering', 'Combined 0 and 1']

# Plot the f1-scores with labels
for i in range(len(f1_scores)):
    plt.plot(range(1, len(f1_scores[i]) + 1), f1_scores[i], label=labels[i], marker='o')

# Add labels and title
plt.xlabel('Fold')
```

```
plt.ylabel('F1-score')
plt.title('F1-scores for Different Clusters')
plt.legend()

# Add numerical values as annotations
for i in range(len(f1_scores)):
    for j, score in enumerate(f1_scores[i]):
        plt.annotate(f'{score:.3f}', (j+1, score), ha='center', va='bottom')

# Show the plot
plt.show()
```

ДОДАТОК Д

```
import xgboost as xgb
from xgboost import plot_importance
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X_train['type'] = le.fit_transform(X_train['type'])
X_train['nameOrig'] = le.fit_transform(X_train['nameOrig'])
X_train['nameDest'] = le.fit_transform(X_train['nameDest'])
X_test['type'] = le.fit_transform(X_test['type'])
X_test['nameOrig'] = le.fit_transform(X_test['nameOrig'])
X_test['nameDest'] = le.fit_transform(X_test['nameDest'])
print(X_train.dtypes)
print(X_test.dtypes)
import numpy as np
import skfuzzy as fuzz
from sklearn.datasets import make_blobs
```

```
import matplotlib.pyplot as plt

# Generate sample data
X, _ = make_blobs(n_samples=1000, centers=3, random_state=42)

# Range of k values to test
k_range = range(2, 10)

# List of F-indices
f_indices = []

# Calculate F-index for each k
for k in k_range:
    cntr, u, _, _, _, _, f = fuzz.cluster.cmeans(X.T, k, 2, error=0.005, maxiter=1000, init=None)
    f_indices.append(f)

# Plot the F-index curve
plt.plot(k_range, f_indices, 'bx-')
plt.xlabel('k')
plt.ylabel('F-index')
plt.title('Fuzzy C-means Clustering')
plt.show()

# Find the optimal k
optimal_k = np.argmax(f_indices) + 2  # add 2 because k_range starts from 2

print(f"The optimal number of clusters is {optimal_k}")
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score
from skfuzzy.cluster import cmeans
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Create an instance of LabelEncoder
le = LabelEncoder()

# Fit and transform the categorical labels in y_train
y_train_encoded = le.fit_transform(y_train)

# Transform the categorical labels in y_test
y_test_encoded = le.transform(y_test)

import skfuzzy as fuzz

# Set the number of clusters
n_clusters = 3

# Apply fuzzy C-means clustering to X_train
cntr_train, u_train, _, _, _, _, _ = fuzz.cluster.cmeans(X_train.T, n_clusters, 2, error=0.005, maxiter=1000, init=None)

# Apply the cluster centers to X_test
u_test = fuzz.cluster.cmeans_predict(X_test.T, cntr_train, 2, error=0.005, maxiter=1000)
```

```
# Plot the training data points
fig, ax = plt.subplots()
ax.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=u_train.argmax(axis=0))

# Plot the cluster centers
ax.scatter(cntr_train[:, 0], cntr_train[:, 1], marker='D', s=150, linewidths=3, color='black')
plt.show()
print("u_train shape:", u_train.shape)
print("u_test shape:", u_test[0].shape)
print(u_test[0])
u_test_transposed = u_test[0].T[:, :2]
print("u_test_transposed shape:", u_test_transposed[0].shape)
u_test_transposed
X_test_new = np.hstack((X_test, u_test_transposed))
X_train_new = np.hstack((X_train, u_train.T[:, :2]))
X_test_new
# Filter data points in cluster 0 from X_train_new and y_train_encoded
X_train_cluster_0 = X_train_new[u_train.argmax(axis=0) == 0]
y_train_cluster_0 = y_train_encoded[u_train.argmax(axis=0) == 0]

# Filter data points in cluster 0 from X_test_new and y_test_encoded
X_test_cluster_0 = X_test_new[u_test[0].argmax(axis=0) == 0]
y_test_cluster_0 = y_test_encoded[u_test[0].argmax(axis=0) == 0]
# Filter data points in cluster 1 from X_train_new and y_train_encoded
X_train_cluster_1 = X_train_new[u_train.argmax(axis=0) == 1]
y_train_cluster_1 = y_train_encoded[u_train.argmax(axis=0) == 1]

# Filter data points in cluster 1 from X_test_new and y_test_encoded
```

```
X_test_cluster_1 = X_test_new[u_test[1].argmax(axis=0) == 1]
y_test_cluster_1 = y_test_encoded[u_test[1].argmax(axis=0) == 1]
# Filter data points in cluster 2 from X_train_new and y_train_encoded
X_train_cluster_2 = X_train_new[u_train.argmax(axis=0) == 2]
y_train_cluster_2 = y_train_encoded[u_train.argmax(axis=0) == 2]

# Filter data points in cluster 2 from X_test_new and y_test_encoded
X_test_cluster_2 = X_test_new[u_test[2].argmax(axis=0) == 2]
y_test_cluster_2 = y_test_encoded[u_test[2].argmax(axis=0) == 2]
# create XGBoost classifier object
xgb_model_0 = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model_0.fit(X_train_cluster_0, y_train_cluster_0,
eval_set=[(X_test_cluster_0, y_test_cluster_0)],
        early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred0 = xgb_model_0.predict(X_test_cluster_0)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_cluster_0, y_pred0)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test_cluster_0,
xgb_model_0.predict_proba(X_test_cluster_0))
print("Logloss:", logloss)
```

```
print(confusion_matrix(y_test_cluster_0, y_pred0))
print(classification_report(y_test_cluster_0, y_pred0))

# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_cluster_0, y_pred0))
print("AUC-ROC:", roc_auc_score(y_test_cluster_0, y_pred0))
# create XGBoost classifier object
xgb_model_1 = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model_1.fit(X_train_cluster_1, y_train_cluster_1, eval_set=[(X_test_cluster_1, y_test_cluster_1)],
                early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred1 = xgb_model_1.predict(X_test_cluster_1)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_cluster_1, y_pred1)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test_cluster_1, xgb_model_1.predict_proba(X_test_cluster_1))
print("Logloss:", logloss)

print(confusion_matrix(y_test_cluster_1, y_pred1))
print(classification_report(y_test_cluster_1, y_pred1))
```

```
# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_cluster_1, y_pred1))
print("AUC-ROC:", roc_auc_score(y_test_cluster_1, y_pred1))
# create XGBoost classifier object
xgb_model_2 = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model_2.fit(X_train_cluster_2, y_train_cluster_2, eval_set=[(X_test_cluster_2, y_test_cluster_2)],
                early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred2 = xgb_model_2.predict(X_test_cluster_2)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_cluster_2, y_pred2)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test_cluster_2, xgb_model_2.predict_proba(X_test_cluster_2))
print("Logloss:", logloss)

print(confusion_matrix(y_test_cluster_2, y_pred2))
print(classification_report(y_test_cluster_2, y_pred2))

# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_cluster_2, y_pred2))
print("AUC-ROC:", roc_auc_score(y_test_cluster_2, y_pred2))
```

```
# create XGBoost classifier object
xgb_model_new = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model_new.fit(X_train_new, y_train_encoded, eval_set=[(X_test_new, y_test_encoded)],
                  early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred_new = xgb_model_new.predict(X_test_new)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test_encoded, y_pred_new)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test_encoded, xgb_model_new.predict_proba(X_test_new))
print("Logloss:", logloss)

print(confusion_matrix(y_test_encoded, y_pred_new))
print(classification_report(y_test_encoded, y_pred_new))

# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test_encoded, y_pred_new))
print("AUC-ROC:", roc_auc_score(y_test_encoded, y_pred_new))
from xgboost import XGBClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import classification_report, f1_score
# create XGBoost classifier object
```

```
xgb_model = XGBClassifier(objective='binary:logistic')

# fit the model on training data with validation set specified
xgb_model.fit(X_train, y_train, eval_set=[(X_test, y_test)],
              early_stopping_rounds=10, verbose=1)

# Make predictions on the clustered test data
y_pred = xgb_model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print the logloss
logloss = log_loss(y_test_encoded, xgb_model.predict_proba(X_test))
print("Logloss:", logloss)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
# Оценка качества модели на тестовых данных
print("F1-score:", f1_score(y_test, y_pred))
print("AUC-ROC:", roc_auc_score(y_test, y_pred))
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6,8)
xgb.plot_importance(xgb_model_0)
```