

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему Аналіз методів збільшення training dataset для табличних даних

Виконав: студент 2 курсу, групи МФ-61
спеціальність 122 «Комп'ютерні
науки»
освітньо-наукова програма
«Інформатика»

Семенов Вадим Костянтинович
(прізвище та ініціали)

Керівник Морозова Анастасія Геннадіївна
(прізвище та ініціали)

Рецензент д.т.н., доцент Давидов В.В
(прізвище та ініціали)

Харків – 2023 року

1. ВСТУП

- 1.1. Формулювання мети роботи, задач та обґрунтування актуальності теми
- 1.2. Стислий огляд відомих результатів
- 1.3. Відомості про одержані результати
- 1.4. Теоретичне та практичне значення результатів, можливі області використання, результати

2. ОСНОВНА ЧАСТИНА

- 2.1. Постановка задачі
- 2.2. Описання та обґрунтування алгоритмів та результатів дослідження

3. ВИСНОВКИ

4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ВСТУП

1.1. Формулювання мети роботи, задач та обґрунтування актуальності теми

Метою дипломної роботи було проаналізувати методи та алгоритми для збільшення training dataset для табличних даних.

Для досягнення поставленої мети було сформульовано наступні задачі:

1. Проаналізувати наявні бібліотеки для збільшення табличних даних та обрати одну з них.
2. Проаналізувати методи роботи бібліотеки з даними та за можливості покращити якість роботи алгоритму з ними, підібравши найкращі параметри для роботи алгоритму. А саме:
 - Скалери
 - Алгоритми оптимізації для моделі глибокого навчання
 - Інші функції бібліотек sklearn, torch, pandas
3. За можливості вдосконалити роботу бібліотеки для роботи з типами даних string, int.
4. Протестувати роботу бібліотеки на датасетах з різними видами розподілу. У якості датасету використовувати реальні дані, а не згенеровані.

Дана тема є актуальною у сьогодення через високу популярність штучного інтелекту та глибинного навчання. А також прогнозується зростання популярності на 20.5% у 2023-2024 роках. [1] Використання моделі, яка могла б збільшити обсяг training dataset могло б стати у нагоді компаніям, що використовують у своїх дослідженнях роботу з табличними

даними та навчають на них свою модель штучного інтелекту. Тому що використання штучно згенерованих даних вирішує такі проблеми як:

- Необхідність великої кількості досліджень та збору даних (опитування користувачів сервісу, проведення тестових випробувань, тощо)
- Аналіз зібраних даних
- Відкидання неправдивих даних (навмання заповнені бюлетені, помилки через технічні причини, тощо)
- Оцифрування даних

1.2. Стислий огляд відомих результатів

Бібліотеку `deep_tabular_augmentation` дійсно можна використовувати для генерування штучних даних, але вона все ж має недоліки [2][3] і її не можна використовувати для будь яких даних.

1.3. Відомості про одержані результати

У результаті моєї праці було:

1. Адаптовано бібліотеку `deep_tabular_augmentation` для роботи зі стовпчиками, що мають тип даних `String` та покращено роботу з типом даних `Int`.
2. Протестовано різні параметри що впливають на генерацію даних та знайдено найбільш оптимальні значення.

3. Протестовано роботу з різними датасетами, що мають різні типи розподілу даних. Датасети були взяті з сайту <https://www.kaggle.com>, а отже це не штучно згенеровані дані та вони відповідають тим, що можуть використовуватись у реальних проектах.

1.4. Теоретичне та практичне значення результатів, можливі області використання, результати

Теоретично, робота цього модулю може бути в нагоді будь-якій компанії, яка застосовує у своїх дослідженнях штучний інтелект та табличні дані. Він має зменшити витрати на збір даних тим самим пришвидшити впровадження нових змін на проєкті. А також покращити якість навчання моделі.

На практиці цей алгоритм не можна використовувати для будь-яких проєктів в незалежності від області в якій вона має використовуватись та даних з якими вона повинна працювати. Але у випадку, якщо мета проєкту відповідає можливостям алгоритму, він може стати незамінним інструментом у роботі в проєкті.

Модуль може бути корисним в будь-якому проєкті, що спеціалізується на роботі з табличними даними та штучним інтелектом, та особливо у випадках, коли є необхідність у великій кількості досліджень та збору даних, але їх за якоїсь причини важко зібрати у необхідній кількості.

Алгоритм здатний генерувати нові дані базуючись на вхідному датасеті. Він може працювати з невеликою кількістю вхідних рядків та генерувати задану кількість нових. Нові рядки базуються здебільшого на середньому значенні вхідних рядків та копіюють їх тип розподілу відносно

інших стовпчиків таблиці. Тим самим згенеровані дані візуально здаються схожими на ті, на яких проводилось тренування моделі.

2. ОСНОВНА ЧАСТИНА

2.1. Постановка задачі

Data augmentation (збільшення даних) — це техніка штучного створення нових даних з наявних даних і значного збільшення різноманітності даних, доступних для навчання моделей. Це робиться шляхом застосування предметно-орієнтованих методів до підмножини навчальних даних. Оскільки продуктивність моделі сильно залежить від якості та кількості набору даних, використання синтетично згенерованих даних може певною мірою допомогти покращити продуктивність моделі. [4]

Для досягнення поставленої мети моєї роботи необхідно проаналізувати наявні методи збільшення табличних даних, розроблені бібліотеки та обрати одну з них. Після вибору бібліотеки необхідно проаналізувати методи, які використовуються модулем, їх недоліки, сильні та слабкі сторони. За можливості слабкі сторони треба вдосконалити або задокументувати та знайти методи їх уникнення. Для тестування роботи алгоритму використовувати дані, отримані з реальних датасетів, а не штучно згенеровані.

2.2. Описання та обґрунтування алгоритмів та результатів дослідження

Підготовка до використання алгоритму data augmentation з використанням бібліотеки deep_tabular_augmentation складається з декількох етапів:

1. Підготовка та масштабування ознак.
2. Розбиття даних.
3. Визначення топології нейронної мережі
4. Вибір оптимізатора
5. Визначення кількості епох для навчання моделі

Масштабування ознак.

Перший етап - це Масштабування ознак (або Нормалізація даних). Перш за все датасет необхідно підготувати для роботи з ним. Для цього використовують масштабування ознак. Так як значення у даних можуть сильно різнитися між собою та мати різні діапазони, модель може давати хибні результати. Тому дані потрібно нормалізувати. Для цього використовують різні скалери з sklearn. В залежності від типу розподілу даних необхідно обрати відповідний скалер. Наприклад, MinMaxScaler та StandardScaler гарно працюють з числовими даними. Водночас StandardScaler використовують для нормального розподілу, MinMaxScaler за відсутності нормального розподілу та коли варто вказати на чітку відстань між значеннями. У моїх дослідженнях для більшості датасетів якості роботи StandardScaler достатньо. Варто також зазначити, що деякі скалери, наприклад Normalizer неможливо використовувати з бібліотекою deep_tabular_augmentation через те, що розробник не впровадив необхідні для цього зміни в свій модуль.

табл.1 Порівняння скалерів

Скалер	Стійкий до викидів	Краще працює з чіткою межею даних	Краще працює, коли межа невідома
StandardScaler	-	-	+
MinMaxScaler	-	+	-
MaxAbsScaler	+	+	-
RobustScaler	+	-	+
PowerTransformer	+-	+-	+

Розбиття даних.

Наступним етап є Розбиття даних. дані розбиваються на дві частини: для тренування та валідації моделі. Попередньо рядки перемішують між собою. Таким чином дані для валідації використовуються для того щоб зрозуміти наскільки навчена модель, а також це допомагає виявити проблеми Underfitting та Overfitting. Зазвичай для розбиття даних використовується `train_test_split()` з бібліотеки `sklearn.model_selection` але для більш специфічних завдань можна звернути увагу на `split_df()` з `mlprepare`. Низькі значення функції втрат можуть вказувати на те, що модель гарно навчилася генерувати нові дані, або на те, що вона перенавчена. У моєму випадку найнижчі значення функції втрат, при яких зберігається залежність між даними, та модель не перенавчається, досягаються при виділенні десяти відсотків даних на валідацію.

Топологія нейронної мережі.

Після масштабування ознак необхідно вказати, за допомогою якої топології нейронної мережі будуть опрацьовуватися дані. Ця топологія

вказує на зв'язки між вузлами (нейронами) в мережі. Для вирішення моєї задачі найбільш цікавими є наступні три типи мережі:

- Auto Encoder (AE)
- Variational AE (VAE)
- Sparse AE (SAE)

Загалом вони використовуються для класифікації та кластеризації ознак. VAE на відміну від AE приділяє більше уваги на зв'язок між даними у той час коли AE намагається їх узагальнити. SAE схожий на VAE, але також здатний знаходити приховані шаблони групування даних. На практиці це виявляється у тому, що SAE виділяє значно більше даних, що знаходяться значно далі від основного скупчення. Наприклад, якщо взяти нормальний розподіл, то SAE буде також виділяти точки, що знаходяться біля 0, у порівнянні з AE та VAE, котрі виділяють лише дані близькі до середнього значення. Тож я вирішив зупинитися на SAE тому що для моєї задачі важливо вказати всі дані, а не лише близькі до середніх значень.

Найближчі до початкових даних результати роботи алгоритму я зміг отримати, коли слої топології виглядають наступним чином: 6, 20, 6. Тобто 6 нейронів у вхідному і вихідному слої та 20 у прихованому. При цьому ці зміни майже не впливають на зміну середнього значення чи дисперсії даних, а змінюється здебільшого лише залежність даних між собою та вірогідність отримати мінімальних та максимальних значень одночасно в декількох стовпчиках таблиці.

Також кількість нейронів має сенс змінювати в залежності від вхідних даних. Емпірично я виявив, що збільшення 1 та 3 слою підвищує "влучність" алгоритму, згенеровані дані стають ближчими до середніх значень розподілу. Збільшення / зменшення прихованого (другого) слою відповідно збільшує/зменшення кількість даних, що знаходяться на значному віддаленні від середнього значення. А ось зменшення першого та третього слою майже не має практичного сенсу, у більшості випадків на

різних даних алгоритм все частіше дає похибки та помилково визначає залежність між розподілом даних. Наприклад, коли розподіл даних схожий на графік функції $y = a^x$ алгоритм може надати $y = \frac{1}{x}$

Оптимізатор навчання нейронних мереж.

Наступний етап підготовки до data augmentation – це обрання алгоритму оптимізації навчання нейронної мережі, який використовується для налаштування ваги нейронної мережі в процесі її навчання. Він визначає, які значення ваги потрібно використовувати для мінімізації функції втрат, яка вимірює помилку передбачення моделі на навчальному наборі даних. Функція втрат – це функція, яка характеризує втрати при неправильному прийнятті рішень на основі спостережених даних. Тобто це метод оцінки того, наскільки добре алгоритм моделює вказаний набір даних, наскільки гарно алгоритм працює з заданим набором. [6]

Найкраще у моїх дослідженнях себе проявили оптимізатори Adam (Адаптивне оцінювання моментів) та RMSProp (Пропагація кореня середньоквадратичного значення), та зовсім погано SGD (Стохастичний градієнтний спуск), незважаючи на його високу популярність. Хоча й Adam більш обчислювально складніший, ніж RMSprop, через необхідність обчислення додаткових моментів градієнта, але дані, що генеруються з його допомогою у більшості випадках більш схожі на початкові у порівнянні з RMSProp. А саме min, max значення ближче до вхідних даних, залежність між даними більш схожа на залежність вхідних даних та менша ймовірність помилково вказати хибну залежність між даними.

Кількість епох при навчанні моделі.

Визначення цього параметру значною мірою залежить від самого датасету. У всіх випадках, у моїх дослідженнях найкращі результати

отримувались при значенні 100-300 епох. При збільшенні цього значення модель починає перенавчатися та згенеровані нею дані все більше дублюють середнє значення початкового датасету, при зменшенні зростає ймовірність похибки алгоритму та неправильно виявленої залежності між даними.

Вдосконалення алгоритму при роботі з типами даних `string` та `int`.

Кластеризація, або кластерний аналіз — це статистична процедура, задача якої полягає в розбитті вибірки об'єктів на підмножини, що не перетинаються і називаються кластерами. [7] Отже типовою задачею кластеризації є розбиття даних на основі їх подібності. На диво, бібліотека `deep_tabular_augmentation` не вміє працювати зі строковими типами даних, хоча якщо строкові дані розбити на невелику кількість кластерів, то виходить, що генерувати нові дані спираючись на порядковий номер кластеру має сенс. Звісно, що такими діями не вийде згенерувати новий текст, а можна лише використовувати старий, а також такі дії будуть мати сенс лише у випадках коли кількість даних більша за кількість кластерів. Тож я вирішив додати алгоритму можливість генерувати дані зі строковим типом та ця ідея виправдала себе.

Також я помітив, що `deep_tabular_augmentation` завжди генерує дані з плаваючою точкою, навіть якщо вхідні дані мають тип `int` та виправив цю помилку.

Види розподілу початкових даних та результати роботи алгоритму з ними.

Мною були протестовані різні датасети, отримані із реальних даних із сайту `kaggle.com`. Нижче представлені результати роботи модулю на деяких із цікавих розподілів та посилання на датасети до них.

Приклад 1. Нормальний розподіл:

<https://www.kaggle.com/datasets/arnabchaki/indian-restaurants-2023>

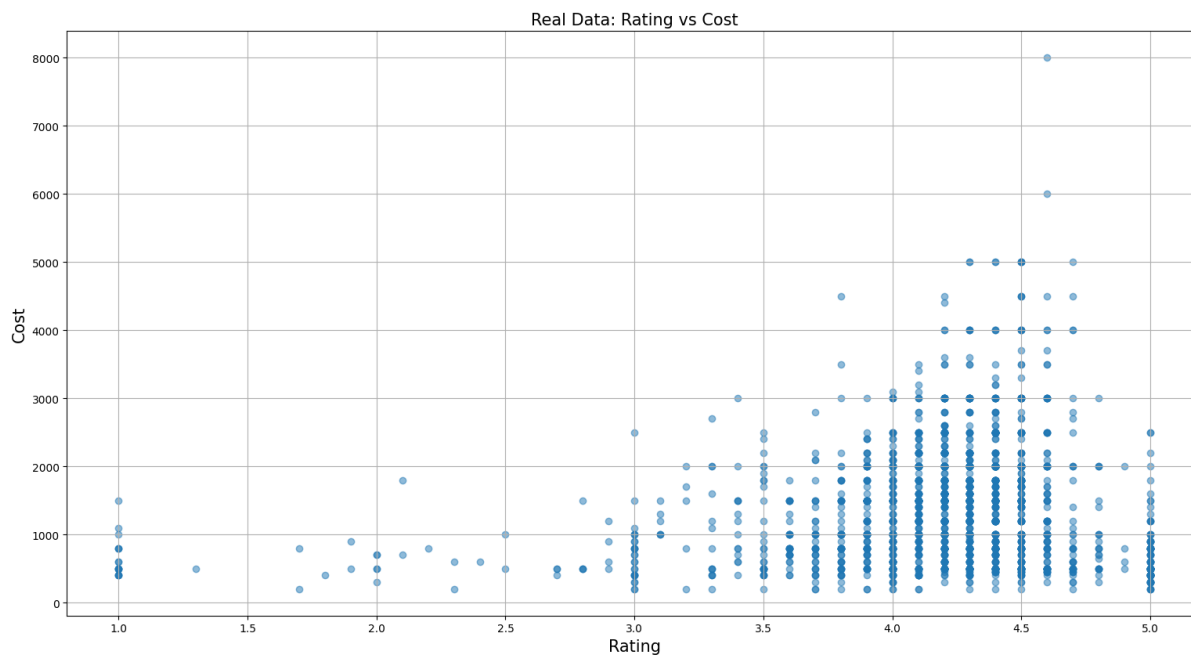


рис.1 Вхідні дані

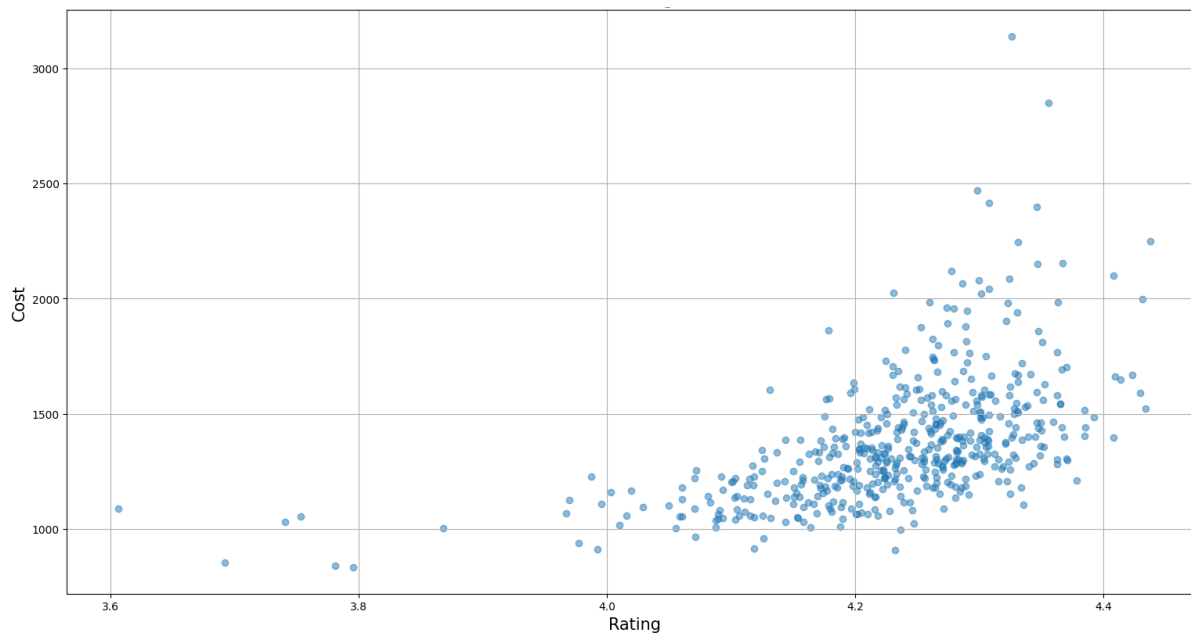


рис.2 Згенеровані дані

табл.2,3 Середнє значення, дисперсія, мінімальне та максимальне значення для вхідних даних та згенерованих

1 df.describe()			1 df_fake_with_noise.describe()		
	Rating	Cost		Rating	Cost
count	2057.000000	2057.000000	count	500.000000	500.000000
mean	4.186582	1396.329606	mean	4.194202	1398.093392
std	0.521078	840.691994	std	0.098244	274.191736
min	1.000000	200.000000	min	3.789653	769.457063
25%	4.000000	800.000000	25%	4.136723	1228.806893
50%	4.300000	1200.000000	50%	4.204616	1354.453507
75%	4.400000	2000.000000	75%	4.265323	1526.335663
max	5.000000	8000.000000	max	4.417704	3301.975625

табл.4 Вхідні дані

1 df.head()									
	Rating	Cost	Name	Location	Locality	City	Cuisine	Votes	Cost.1
0	4.1	2000	Local	Scindia House,Connaught Place, Central Delhi	Central Delhi	Delhi	North Indian, Finger Food, Continental	2415	2000
1	4.3	1500	The G.T. ROAD	M-Block,Connaught Place, Central Delhi	Central Delhi	Delhi	North Indian	2363	1500
2	4.2	2000	Tamasha	Connaught Place, Central Delhi	Central Delhi	Delhi	Finger Food, North Indian, Italian, Contine...	5016	2000
3	4.2	1800	The Junkyard Cafe	Connaught Place, Central Delhi	Central Delhi	Delhi	North Indian, Mediterranean, Asian, Italian...	2821	1800
4	4.4	2000	Chilr's American Grill and Bar	M-Block,Connaught Place, Central Delhi	Central Delhi	Delhi	Mexican, American, Italian	1094	2000

табл.5 Згенеровані дані

1 df_fake_with_noise.head()									
	Rating	Cost	Name	Location	Locality	City	Cuisine	Votes	Cost.1
0	4.173700	1482.0	Gallery Cafe	Sector 21C, Faridabad	Central Suburbs	Ghaziabad	North Indian, European	228.0	1431.0
1	4.232538	1410.0	Ambarsari Chowk	Narela, North Delhi	Central Suburbs	Ghaziabad	Multi-Cuisine, North Indian, Italian, Chine...	92.0	1377.0
2	4.194687	1431.0	Cheese And More - The Kitchen	Radisson Hotel Gurugram Sohna Road City Center...	NCR Greater Noida	Gurgaon	Asian, Chinese, Thai	213.0	1230.0
3	4.129451	1411.0	Mogli's Coffee	DLF Star Mall,Sector 30, Gurgaon	NCR Greater Noida	Ghaziabad	North Indian, Chinese, Mughlai, Barbecue	220.0	1346.0
4	4.176762	1357.0	Nathu's Sweets	Clarens Hotel,Sector 29, Gurgaon	Faridabad	Ghaziabad	Multi-Cuisine, Italian, Chinese, Continental	190.0	1529.0

Приклад 2. Пряма

<https://www.kaggle.com/datasets/rajkumarpandey02/tesla-inc-tsla-stock-price>

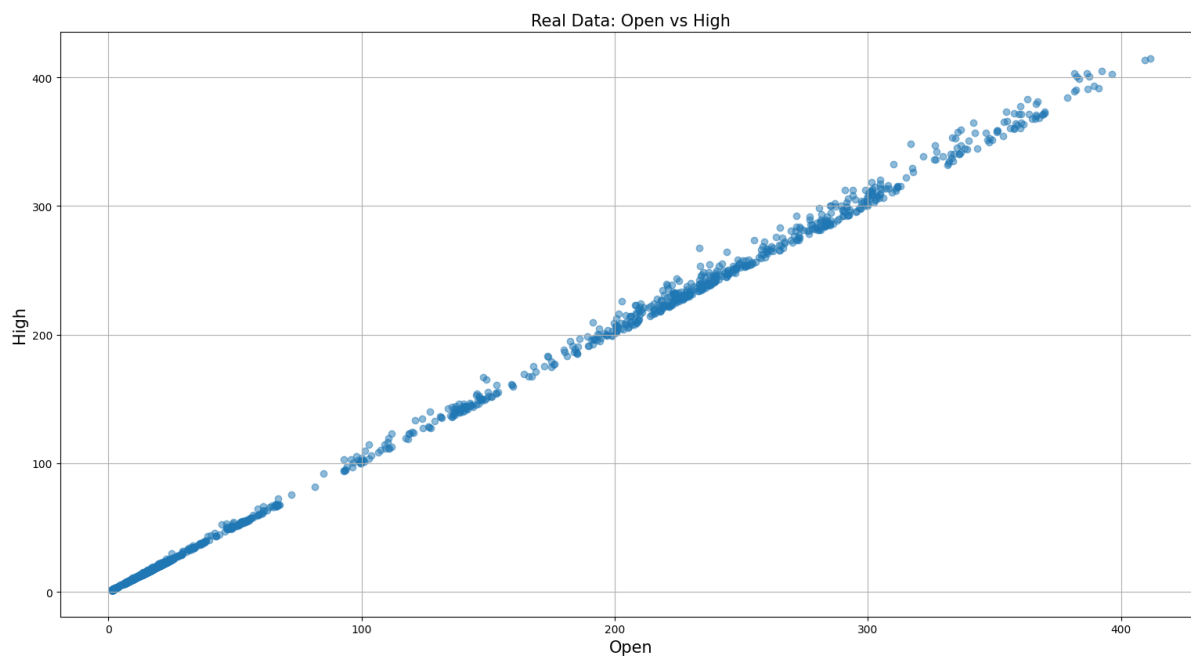


рис.3 Вхідні дані

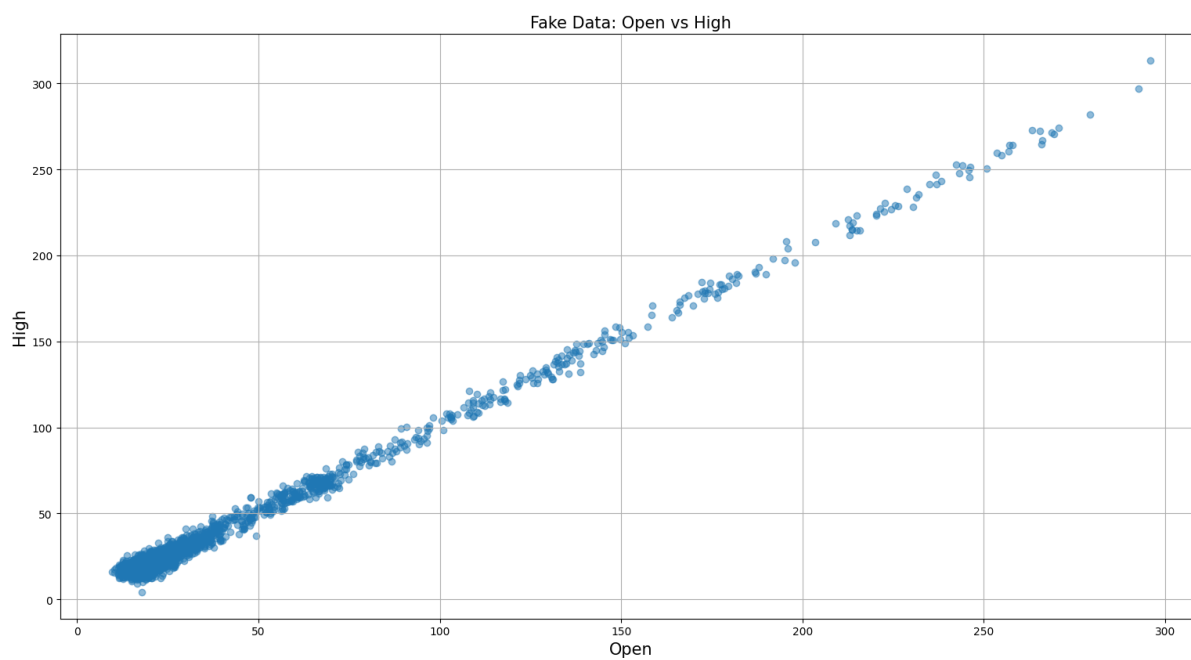


рис.4 Згенеровані дані

табл.6 Середнє значення, дисперсія, мінімальне та максимальне значення для вхідних даних

```
1 df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	3024.000000	3024.000000	3024.000000	3024.000000	3024.000000	3.024000e+03
mean	61.391878	62.767023	59.869111	61.338114	61.338114	9.673891e+07
std	96.891156	99.117871	94.372086	96.757037	96.757037	8.174686e+07
min	1.452000	1.484667	1.407333	1.455333	1.455333	3.594000e+06
25%	11.516667	11.859000	11.212500	11.562500	11.562500	4.700618e+07
50%	16.611999	16.852333	16.380667	16.597333	16.597333	7.826100e+07
75%	29.053333	30.111333	28.391333	28.701333	28.701333	1.198309e+08
max	411.470001	414.496674	405.666656	409.970001	409.970001	9.140820e+08

табл.7 Середнє значення, дисперсія, мінімальне та максимальне значення для згенерованих даних

```
1 df_fake_with_noise.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2.000000e+03
mean	42.066105	43.160800	41.038195	42.081923	42.041818	9.886634e+07
std	47.009153	47.993177	45.852147	46.907366	46.866173	3.409333e+06
min	9.598818	4.323151	10.670437	8.866035	9.561628	8.231404e+07
25%	17.860082	18.362353	17.478084	17.885111	18.067284	9.672895e+07
50%	22.313836	23.000097	21.648599	22.307078	22.500258	9.865850e+07
75%	40.155833	42.715501	40.534891	41.559376	40.471391	1.008191e+08
max	295.823052	313.146319	291.140474	299.601484	300.061823	1.112912e+08

табл.8 Вхідні дані

```
1 df.head()
```

	Open	High	Low	Close	Adj Close	Volume
0	1.789333	1.800000	1.726667	1.774667	1.774667	19245000
1	1.777333	1.796667	1.734667	1.778000	1.778000	17811000
2	1.765333	1.793333	1.746000	1.788667	1.788667	21700500
3	1.788667	1.866667	1.787333	1.858667	1.858667	30918000
4	1.866667	1.905333	1.860000	1.882667	1.882667	33718500

табл.9 Згенеровані дані

```
1 df_fake_with_noise.head()
```

	Open	High	Low	Close	Adj Close	Volume
0	17.262359	16.285746	17.428413	13.738211	18.802615	96473735.0
1	21.177621	24.055063	21.155177	23.533396	24.248490	98946420.0
2	16.903359	18.169311	21.098675	19.451377	18.497988	99666311.0
3	39.148747	37.669324	35.729360	34.946729	32.419110	102750475.0
4	96.361363	99.684192	90.139890	96.907740	94.742329	107642622.0

Приклад 3. Кластери

<https://www.kaggle.com/datasets/anandaramg/luxury-loan-portfolio>

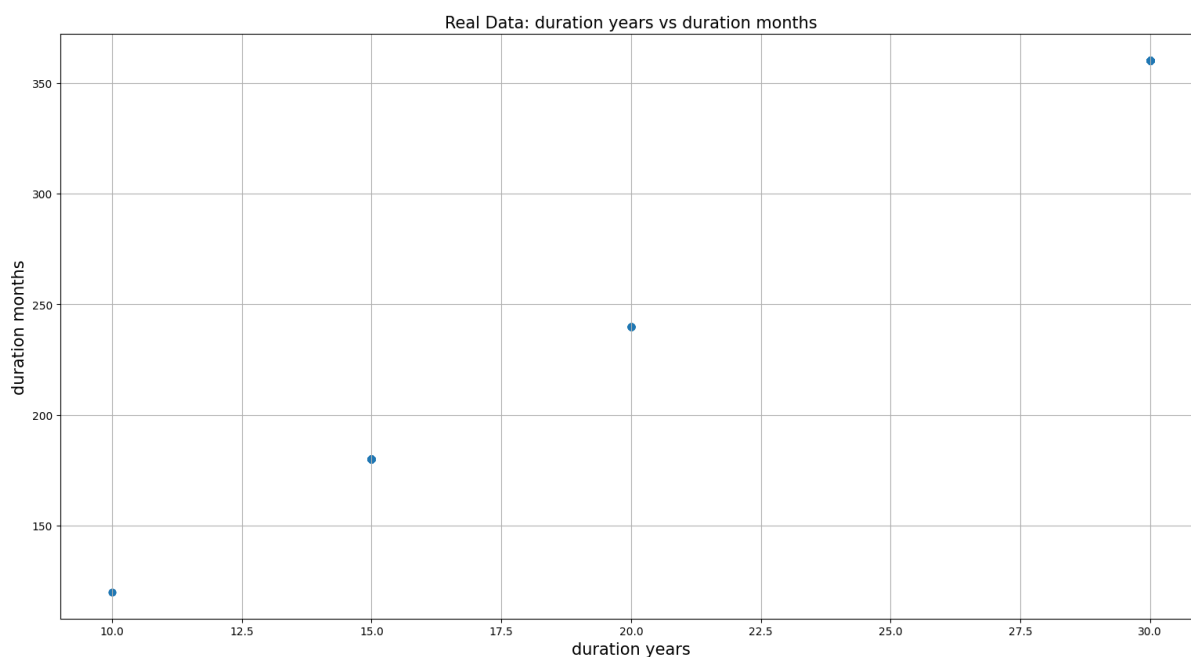


рис.5 Вхідні дані

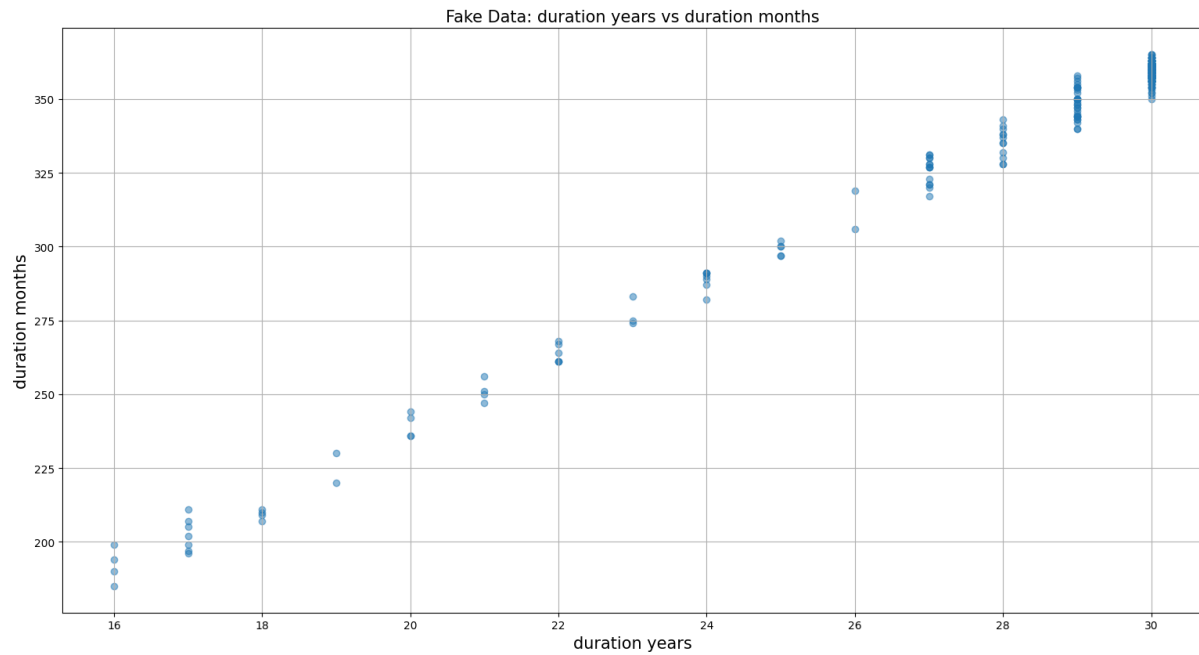


рис.6 Згенеровані дані

табл.10,11 Середнє значення, дисперсія, мінімальне та максимальне значення для вхідних даних та згенерованих

1 df.describe()		
	duration years	duration months
count	1678.000000	1678.000000
mean	27.362932	328.355185
std	5.616759	67.401114
min	10.000000	120.000000
25%	30.000000	360.000000
50%	30.000000	360.000000
75%	30.000000	360.000000
max	30.000000	360.000000

1 df_fake_with_noise.describe()		
	duration years	duration months
count	500.000000	500.000000
mean	28.932000	346.772000
std	2.860724	34.631412
min	16.000000	185.000000
25%	30.000000	356.000000
50%	30.000000	359.000000
75%	30.000000	361.000000
max	30.000000	365.000000

табл.12,13 Вхідні дані та згенеровані дані

1 df.head()			1 df_fake_with_noise.head()		
	duration	years	duration	years	months
0		20	30.0		359.0
1		15	29.0		352.0
2		15	30.0		361.0
3		20	30.0		361.0
4		20	27.0		323.0

Приклад 4. Приклад зі сторінки розробника [8]



рис.7 Вхідні дані

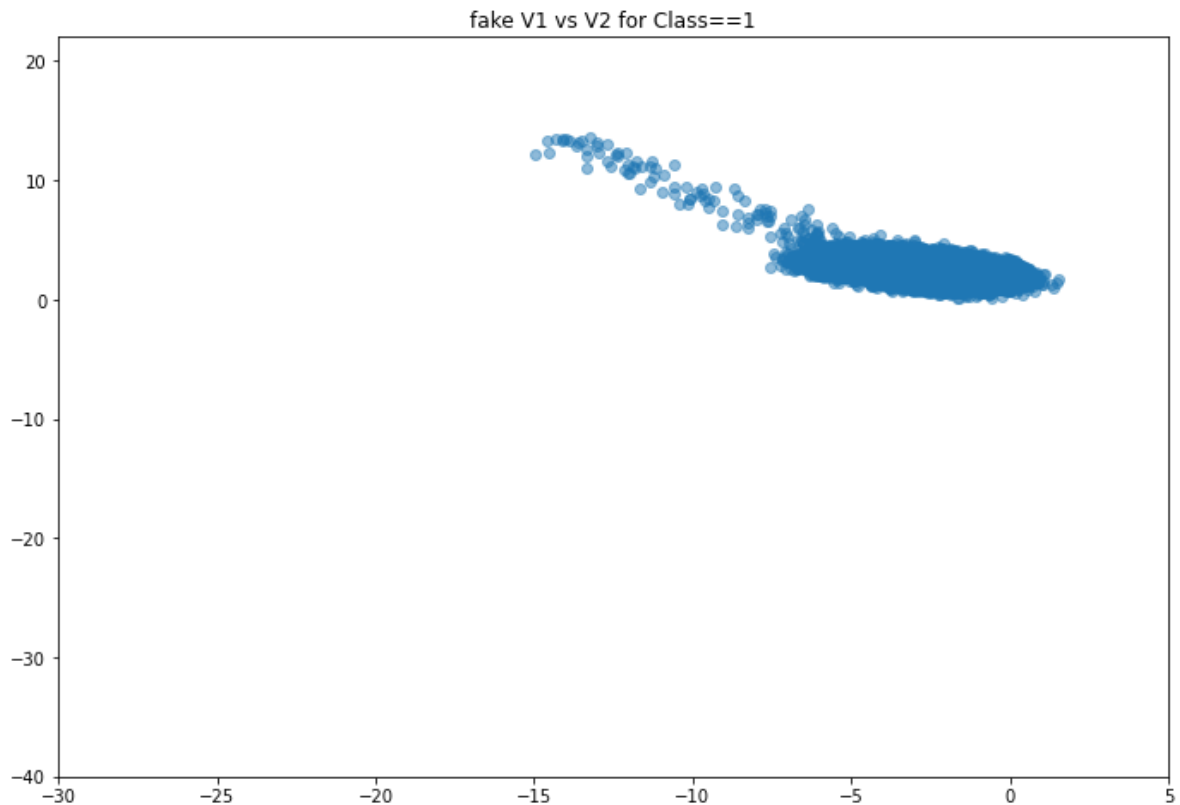


рис.8 Згенеровані дані

табл.14,15 Середнє значення, дисперсія, мінімальне та максимальне значення для вхідних даних та згенерованих

df[df['Class']==1].describe()						df_fake_with_noise.describe()					
	Time	V1	V2	V3	V4		Time	V1	V2	V3	V4
count	492.000000	492.000000	492.000000	492.000000	492.000000	count	283823.000000	283823.000000	283823.000000	283823.000000	283823.000000
mean	80746.806911	-4.771948	3.623778	-7.033281	4.542029	mean	86350.536365	-2.730339	2.526086	-4.936909	4.129437
std	47835.365138	6.783687	4.291216	7.110937	2.873318	std	7553.990281	1.086138	0.571059	0.936746	0.424873
min	406.000000	-30.552380	-8.402154	-31.103685	-1.313275	min	47336.969554	-14.942467	0.089373	-11.987048	2.412368
25%	41241.500000	-6.036063	1.188226	-8.643489	2.373050	25%	82525.876360	-3.392543	2.143453	-5.549135	3.837401
50%	75568.500000	-2.342497	2.717869	-5.075257	4.177147	50%	87472.050892	-2.628975	2.509311	-4.904040	4.093337
75%	128483.000000	-0.419200	4.971257	-2.276185	6.348729	75%	91465.706327	-1.977823	2.890840	-4.296778	4.384297
max	170348.000000	2.132386	22.057729	2.250210	12.114672	max	112896.524300	1.551142	13.593776	-1.207216	6.155796

табл.16,17 Вхідні дані та згенеровані дані

df.head()						df_fake_with_noise.head()					
	Time	V1	V2	V3	V4		Time	V1	V2	V3	V4
0	0.0	-1.359807	-0.072781	2.536347	1.378155	0	88337.983697	-2.048481	2.552722	-4.830715	3.785770
1	0.0	1.191857	0.266151	0.166480	0.448154	1	90035.749837	-2.367102	2.463058	-5.169741	4.054593
2	1.0	-1.358354	-1.340163	1.773209	0.379780	2	87295.225013	-2.118643	2.813614	-4.153329	3.675768
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	3	88781.785086	-2.084409	2.471753	-4.923555	4.004303
4	2.0	-1.158233	0.877737	1.548718	0.403034	4	86419.344641	-3.918795	1.493713	-4.635602	4.038000

Недоліки бібліотеки `deep_tabular_augmentation`.

Найбільш суттєвим недоліком є, на мою думку, майже повна відсутність дисперсії у згенерованих даних. Дані дійсно мають схоже середнє значення, але наприклад, коли вхідні дані мають нормальний розподіл, згенеровані візуально більше схожі на криву лінію, чи в деяких випадках дві криві лінії, що перетинаються.

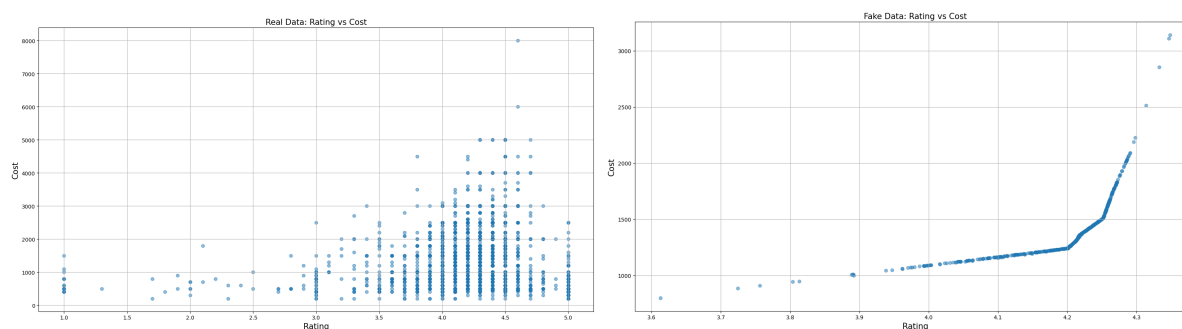


рис.9, 10 Вхідні дані, згенеровані дані

Для вирішення цієї проблеми розробник радить трохи зміщувати дані, на відстань рівну 10% від дисперсії. У результаті ми отримуємо приблизно наступний графік.

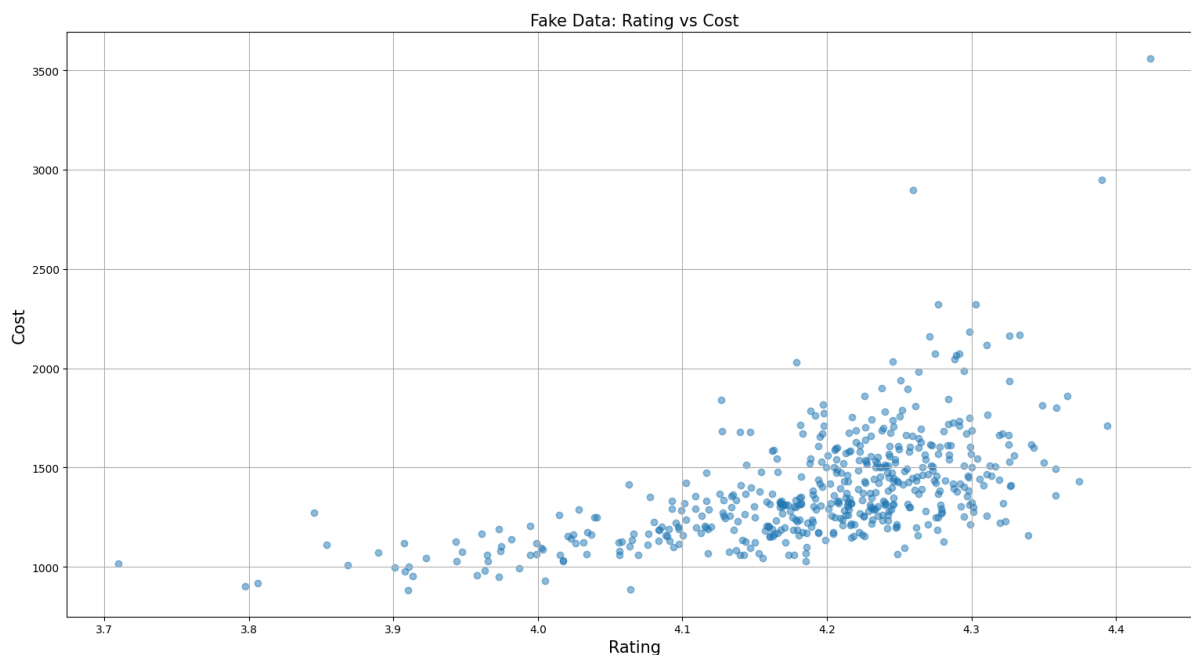


рис.11 Згенеровані дані зі зміщенням 10% від дисперсії

Звісно, що в результаті ми ніколи не матимемо однакові дисперсії, а в спробі змістити дані на відстань рівну дисперсії ми можемо отримати у чисельному вигляді майже однакові середні значення та дисперсію, у порівнянні з початковими даними, але візуально це вже зовсім не схоже на нормальний розподіл, тобто залежність між даними втрачається.

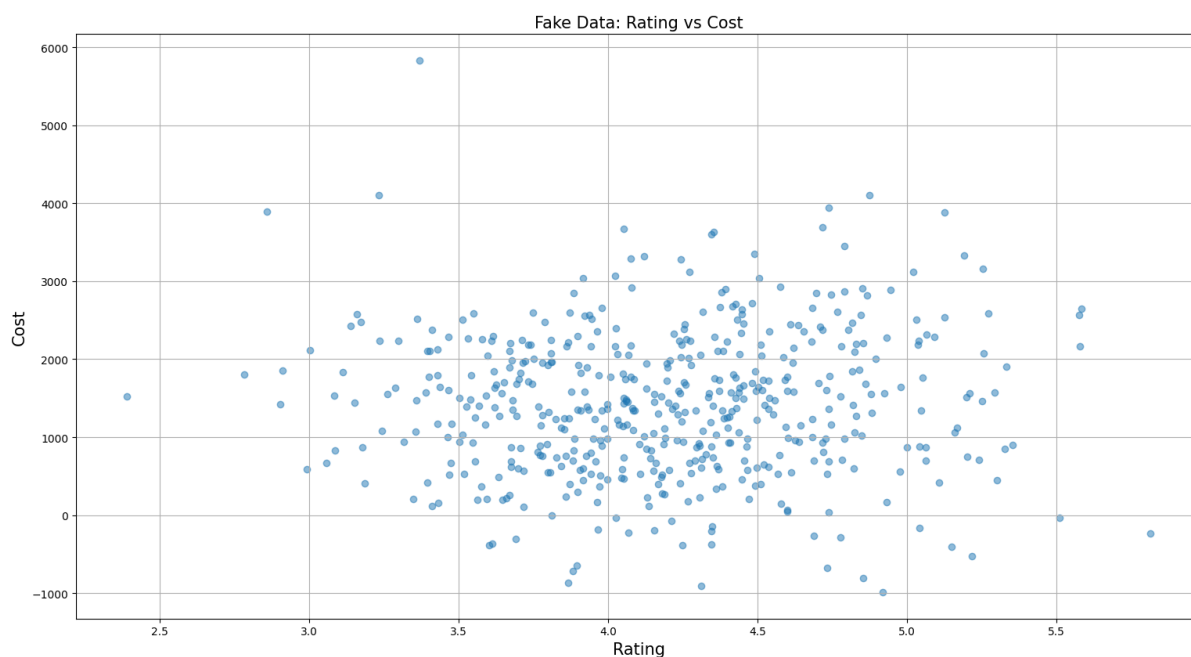


рис.12 Згенеровані дані зі зміщенням 100% від дисперсії

Чисельне порівняння:

табл.18,19,20 Середнє значення та дисперсія вхідних даних, згенерованих без зміщення та згенерованих зі зміщенням 10% від дисперсії

1 df.describe()			1 df_fake.describe()			1 df_fake_with_noise.describe()		
	Rating	Cost		Rating	Cost		Rating	Cost
count	2057.000000	2057.000000	count	500.000000	500.000000	count	500.000000	500.000000
mean	4.186582	1396.329606	mean	4.189291	1386.404053	mean	4.192461	1406.085324
std	0.521078	840.691994	std	0.090886	271.137085	std	0.096993	303.819154

3. ВИСНОВКИ

Мною було проаналізовано методи та алгоритми для збільшення training dataset для табличних даних. Для досягнення цієї мети я обрав бібліотеку deep_tabular_augmentation та проаналізував функції, що в ній використовуються. Під час аналізу я підібрав діапазони значень вхідних параметрів, при яких досягається найвища точність роботи алгоритму та згенеровані дані найбільше нагадують початкові по критеріям середнього значення, максимального, мінімального та візуально спостережувальна залежність між даними. А саме:

- функція втрат
- Скалери
- Топологія нейронної мережі
- Оптимізатор навчання нейронних мереж
- Визначена оптимальна кількість епох

Також мною були помічені та проаналізовані недоліки модулю та покращена якість роботи алгоритму при роботі з типами даних string, int.

4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. <https://www.ml-science.com/blog/2020/10/8/artificial-neural-networks-popularity>
2. https://github.com/lshmiddey/deep_tabular_augmentation/issues/19
3. https://github.com/lshmiddey/deep_tabular_augmentation/issues/12
4. https://www.mphasis.com/content/dam/mphasis-com/global/en/home/innovation/next-lab/Mphasis_Data-Augmentation-for-Tabular-Data_Whitepaper.pdf
5. https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BE%D0%B7%D0%BD%D0%B0%D0%BA
6. <https://ekmair.ukma.edu.ua/server/api/core/bitstreams/3c1fe679-0e02-4bf6-a5fa-33499b58f6fe/content>
7. http://csc.knu.ua/media/study/asp/mod_probl_inf_tech_sys_analysivs_ivohin/lecture/lec11.pdf
8. https://github.com/lshmiddey/deep_tabular_augmentation/blob/main/Notebooks/DeepLearning_DataAugmentation_RF.ipynb#enroll-beta