Бібліотеки

Pandas.

Pandas - це високорівнева бібліотека Python для аналізу даних. Високорівнева вона тому, що побудована поверх низькорівневої бібліотеки NumPy (написана на С), що є великим плюсом у продуктивності. В екосистемі Python, pandas є найбільш просунутою бібліотекою, що швидко розвивається, для обробки та аналізу даних.

Numpy.

NumPy (скорочено від Numerical Python) — бібліотека з відкритим кодом для мови програмування Python. Можливості:

* підтримка багатовимірних масивів (включаючи матриці);
* підтримка високорівневих математичних функцій, призначених для роботи з багатовимірними масивами.

Matplotlib.

Це інтерфейс на основі стану до matplotlib. Він забезпечує неявний, схожий на MATLAB, спосіб побудови. Він також відкриває графіки на вашому екрані та діє як менеджер графічного інтерфейсу користувача. Pyplot в основному призначений для інтерактивних графіків і простих випадків програмної генерації графіків.

StandardScaler.

StandardScaler із sklearn.preprocessing стандартизує функції шляхом видалення середнього значення та масштабування до одиничної дисперсії. Стандартна оцінка зразка x обчислюється як: z = (x - u) / s

**PCA**

**PCA із sklearn.decomposition -** аналіз головних компонентів (PCA).

Зменшення лінійної розмірності за допомогою декомпозиції за одиничними значеннями для проектування їх у нижчий розмірний простір. Вхідні дані центруються, але не масштабуються для кожної функції перед застосуванням SVD.

Enum

Is a set of symbolic names (members) bound to unique values.

skfuzzy

Цей SciKit — це інструментарій нечіткої логіки для SciPy.

**Train\_test\_split**

**Train\_test\_split зі sklearn.model\_selection -** розбиває масиви або матриці на випадкові тренувальні та тестові підмножини.

KMeans

KMeans зі sklearn.cluster - K-Means кластеризація.

Cdist

Cdist зі scipy.spatial.distance - обчислює відстань між кожною парою з двох наборів вхідних даних.

**Розібратися**

**З параметром stratify в методі def split\_on\_train\_and\_test\_datasets(self, dataframe, age\_bins=False):**

**:**# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
train\_bin**,** test\_bin = train\_test\_split(bin\_data**,** test\_size=test\_size**,** random\_state=**42**)

# Розбиваємо вибірку на навчальний та тестовий набори  
# test\_size=0.1 означає, що 10% даних піде у тестовий набір  
X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.1,** random\_state=**42**)

Для того, щоб розбити вибірку на **навчальний** та **тестовий** набори даних, можна скористатися різними підходами. Один з найпоширеніших методів — це **метод випадкового поділу**. В Python такий поділ можна виконати за допомогою бібліотеки scikit-learn.

**Кроки для розбиття вибірки:**

1. **Визначте вхідні дані та цільову змінну**:
   * **Вхідні дані** (ознаки або біомаркери) — це набір змінних, які використовуються для побудови моделі (наприклад, біомаркери крові).
   * **Цільова змінна** — це значення, яке ви намагаєтесь передбачити (наприклад, біологічний або фактичний вік).
2. **Визначте співвідношення між навчальними та тестовими даними**:
   * Найчастіше використовується співвідношення 80% для **навчального набору** та 20% для **тестового**. Але можливі й інші варіанти (наприклад, 70/30 або 90/10), залежно від задачі та кількості даних.
3. **Випадковий поділ**:
   * Дані діляться випадковим чином, щоб зберегти репрезентативність обох наборів. Це можна зробити за допомогою функції train\_test\_split з бібліотеки scikit-learn.

Припустимо, що у нас є DataFrame з біомаркерами. В колонці "Age" знаходиться цільова змінна (вік), а решта колонок — це ознаки (біомаркери).

**Пояснення:**

* **X** — це вхідні дані (матриця біомаркерів).
* **y** — це цільова змінна (вік).
* **train\_test\_split()** — функція, що ділить дані на навчальний та тестовий набори. Параметр test\_size=0.1 означає, що 10% даних буде виділено для тестування, а решта 90% — для навчання.
* **random\_state=42** — використовується для фіксації випадковості, щоб поділ був відтворюваним при кожному запуску.

**Що робити з навчальним та тестовим наборами?**

* **Навчальний набір**: Використовується для тренування моделі, підбору параметрів і налаштувань.
* **Тестовий набір**: Використовується для оцінки якості та точності моделі на нових даних, які модель не бачила під час навчання.

**Додаткові поради:**

* Якщо ваша вибірка має нерівномірний розподіл цільової змінної (наприклад, дисбаланс вікових груп), ви можете використовувати параметр stratify=y у функції train\_test\_split. Це гарантує, що пропорції різних класів у навчальному і тестовому наборах будуть однакові.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42, stratify=y)

Цей код допоможе вам рівномірно розподілити вікові групи між навчальним і тестовим наборами.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

Що таке дисбаланс вікових груп?

**Програма ChatGPT сказала:**



**Дисбаланс вікових груп** — це ситуація, коли в наборі даних присутня нерівномірна кількість прикладів для різних вікових категорій. Наприклад, якщо в наборі даних набагато більше людей молодшого віку (наприклад, 20-30 років), ніж людей старшого віку (наприклад, 60-70 років), то виникає дисбаланс. Це може негативно вплинути на побудову та точність моделей машинного навчання, які передбачають або класифікують вік.

**Як дисбаланс впливає на моделі?**

1. **Погана генералізація**: Модель буде добре працювати на частих (переважаючих) групах і погано — на рідкісних групах. Якщо у вас багато молодих людей у вибірці, модель може почати надмірно акцентуватися на характеристиках цієї групи і погано передбачати вік для старших людей.
2. **Ухил у бік переважаючої групи**: Модель може бути зміщеною до прогнозування молодших вікових груп, оскільки в навчальному наборі більше прикладів з цієї категорії. Це призведе до некоректних передбачень для менш представлених груп (наприклад, людей старшого віку).

**Як виявити дисбаланс?**

1. **Візуальний огляд**: Можна побудувати гістограму розподілу вікових груп і перевірити, чи однаково представлені різні категорії.

Приклад на Python:

python

Копіювати код

import matplotlib.pyplot as plt

# Припустимо, що у вас є колонка 'Age' з віковими даними

data['Age'].hist(bins=10)

plt.title('Розподіл вікових груп')

plt.xlabel('Вік')

plt.ylabel('Кількість')

plt.show()

1. **Описові статистики**: Порахувати кількість прикладів для кожної вікової групи або інтервалу.

python

Копіювати код

age\_distribution = data['Age'].value\_counts()

print(age\_distribution)

**Як боротися з дисбалансом?**

1. **Стратифікований поділ вибірки**: При розбитті даних на навчальний і тестовий набори варто використовувати параметр stratify у функції train\_test\_split, щоб забезпечити рівномірний розподіл вікових груп у кожному наборі.

python

Копіювати код

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

1. **Оверсемплінг**: Для боротьби з дисбалансом можна штучно збільшити кількість прикладів для недостатньо представлених вікових груп. Один із методів — оверсемплінг, який створює копії зразків з рідкісних груп.
2. **Андерсемплінг**: Це протилежний підхід, який зменшує кількість прикладів у переважаючих групах, щоб зрівняти їх із менш представленими групами.
3. **Вага класів у моделі**: У багатьох моделях можна встановити параметри ваги класів, щоб модель більше "приділяла увагу" рідкісним групам. Це допомагає уникнути того, щоб модель орієнтувалася на більш представлені категорії.

Наприклад, у деяких алгоритмах, таких як **Logistic Regression** або **Random Forest**, є параметр class\_weight='balanced'.

1. **Метрики оцінки моделі**: Для нерівномірно збалансованих даних не завжди підходить метрика **точність (accuracy)**, оскільки модель може показати високу точність, але погано передбачати для рідкісних груп. Тому краще використовувати такі метрики, як **середньозважена точність (weighted accuracy)**, **f1-score** або **ROC-AUC**, які враховують дисбаланс.

**Підсумок**

Дисбаланс вікових груп може значно вплинути на якість побудованої моделі, оскільки модель буде орієнтована на переважні групи. Важливо враховувати цей фактор і використовувати стратегії, які допоможуть коректно працювати з нерівномірними даними.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

Привет!

**Програма ChatGPT сказала:**



Привет! Как могу помочь? :)

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

Что за ошибка The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2. При вводе The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2.?

**Програма ChatGPT сказала:**



Ошибка **"The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2"** возникает, когда вы пытаетесь выполнить **стратифицированное разбиение** данных (например, используя параметр stratify в функции train\_test\_split) на основе целевой переменной yyy, но одна из категорий или классов в этой переменной содержит только один экземпляр.

Стратификация требует, чтобы каждая категория или класс целевой переменной yyy имели хотя бы **две записи**, чтобы можно было корректно разбить данные на несколько частей, сохраняя пропорции каждого класса в обучающем и тестовом наборах.

**Причина ошибки:**

* Один из классов в целевой переменной yyy содержит только один экземпляр, что не позволяет корректно выполнить стратифицированное разбиение.

**Решения проблемы:**

1. **Удаление редких классов (если возможно)**: Если класс с одним экземпляром не является важным для вашей задачи, можно его удалить.

Пример на Python:

python

Копіювати код

# Удаляем строки, где класс встречается только один раз

class\_counts = y.value\_counts()

rare\_classes = class\_counts[class\_counts == 1].index

X\_filtered = X[~y.isin(rare\_classes)]

y\_filtered = y[~y.isin(rare\_classes)]

1. **Объединение редких классов**: Если несколько классов имеют мало данных, можно объединить их в один класс, чтобы улучшить распределение данных.

Пример:

python

Копіювати код

y\_combined = y.copy()

rare\_classes = class\_counts[class\_counts == 1].index

y\_combined[y\_combined.isin(rare\_classes)] = 'Rare\_Class'

1. **Использование простого случайного разбиения без стратификации**: Если стратификация не является обязательной, можно выполнить разбиение без параметра stratify. Это позволит избежать ошибки, хотя стратификация помогает улучшить сбалансированность классов в обучающей и тестовой выборках.

Пример:

python

Копіювати код

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

1. **Добавление данных в редкие классы (если возможно)**: Если вы работаете с недостатком данных в конкретных классах, можно попробовать добавить дополнительные данные для этих классов, либо искусственно увеличить их количество (например, с помощью методов генерации новых данных, таких как SMOTE для увеличения малых классов).

Пример использования SMOTE:

python

Копіювати код

from imblearn.over\_sampling import SMOTE

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

**Заключение:**

Эта ошибка сигнализирует о том, что при стратифицированном разбиении некоторые классы представлены слишком малым количеством экземпляров (менее 2). Решение проблемы зависит от того, как важны редкие классы для вашей задачи. Вы можете либо удалить их, объединить с другими классами, либо выполнить случайное разбиение без стратификации.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

Как использовать train\_test\_split, если я хочу разбить на бины диапазон возрастов датафрейма self.Ages и с каждого бина взять определенный процент для тренировочной и тестовой выборок?

**Програма ChatGPT сказала:**



Чтобы разбить данные на бины возрастов и для каждого бина выделить определенный процент для тренировочной и тестовой выборок, можно использовать следующий подход:

1. **Разбить возрастной диапазон на бины**.
2. **Для каждого бина отдельно применить разбиение на тренировочную и тестовую выборки** с нужными процентами.
3. **Объединить результаты в финальные тренировочный и тестовый наборы**.

**Основные шаги:**

1. **Разбить возрастной диапазон на бины**: Для этого можно использовать функцию pd.cut, которая позволяет разбить значения возраста на интервалы (бины).
2. **Применить разбиение для каждого бина**: Для этого удобно будет использовать цикл для применения train\_test\_split к каждому бину отдельно.
3. **Объединить результаты**: После разбиения наборов на тренировочный и тестовый наборы, их можно собрать в общие DataFrame'ы для тренировочной и тестовой выборки.

**Пример на Python**

python

Копіювати код

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Допустим, у вас есть DataFrame с возрастами в колонке 'Ages' и другими признаками в X

data = pd.DataFrame({

'Ages': [20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80],

'Feature1': range(13),

'Feature2': range(13, 26)

})

# Шаг 1: Разбиваем на возрастные бины

# Задаем возрастные интервалы (бины), например, по 10 лет

bins = [20, 30, 40, 50, 60, 70, 80]

labels = ['20-30', '30-40', '40-50', '50-60', '60-70', '70-80']

data['AgeBin'] = pd.cut(data['Ages'], bins=bins, labels=labels)

# Для хранения тренировочных и тестовых данных

train\_data = pd.DataFrame()

test\_data = pd.DataFrame()

# Шаг 2: Проходим по каждому бину и применяем train\_test\_split

for bin\_label in labels:

# Фильтруем строки по конкретному возрастному бину

bin\_data = data[data['AgeBin'] == bin\_label]

if len(bin\_data) > 1: # Проверяем, чтобы в бине было достаточно данных

# Разбиваем на тренировочную и тестовую выборки (например, 70% на обучение и 30% на тест)

train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=0.3, random\_state=42)

# Добавляем результаты в финальные DataFrame'ы

train\_data = pd.concat([train\_data, train\_bin], axis=0)

test\_data = pd.concat([test\_data, test\_bin], axis=0)

# Шаг 3: Сбрасываем индексы для удобства

train\_data = train\_data.reset\_index(drop=True)

test\_data = test\_data.reset\_index(drop=True)

# Результаты

print("Training data:")

print(train\_data)

print("\nTest data:")

print(test\_data)

**Объяснение кода:**

* **Шаг 1**: Мы разбиваем возрастной диапазон (колонка Ages) на интервалы с помощью функции pd.cut(), создавая новую колонку AgeBin, которая содержит метки бинов.
* **Шаг 2**: Проходим через каждый бин (например, 20-30, 30-40 и т.д.), фильтруем данные для этого бина и применяем функцию train\_test\_split для разбиения на тренировочный и тестовый наборы.
* **Шаг 3**: Объединяем все полученные данные в общие DataFrame'ы для тренировочного и тестового набора.

**Примечание:**

* Если в каком-то бине недостаточно данных (например, только 1 запись), лучше обрабатывать такие случаи отдельно (например, пропустить этот бин или объединить его с соседним).
* Параметр test\_size=0.3 задает, какой процент данных будет отобран в тестовый набор (в данном случае 30%). Этот параметр можно настроить по необходимости.

**Результат:**

Вы получите два DataFrame'а — один для тренировочного набора и один для тестового, в которых пропорции возрастных бинов будут сохранены согласно вашему плану.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

А если у меня для каждого бина сильно различаются количество возрастов?

**Програма ChatGPT сказала:**



Если для каждого возрастного бина сильно различается количество данных (например, один бин содержит гораздо больше наблюдений, чем другой), это может привести к дисбалансу при разделении на тренировочную и тестовую выборки. В таких случаях вы можете применить несколько стратегий для корректного разбиения данных:

**Варианты решения:**

1. **Пропорциональное разбиение с учётом размера бина**:
   * Если один бин содержит намного больше данных, чем другие, вы можете сделать пропорциональное разбиение, где процент тренировочных и тестовых данных будет зависеть от количества наблюдений в бине.
2. **Минимальная доля данных для каждого бина**:
   * Для очень маленьких бинов можно использовать минимальную долю данных, например, чтобы гарантировать, что в тестовой выборке будут хотя бы несколько записей для каждой возрастной категории.
3. **Использование методов стратификации**:
   * Вместо разбиения данных по каждому бину отдельно, можно использовать стратификацию на уровне всей выборки, чтобы сохранить пропорции возрастных категорий в тренировочном и тестовом наборах.

**Пример 1: Пропорциональное разбиение с учётом размера бина**

В этом примере мы будем использовать пропорциональное разбиение для каждого возрастного бина, основываясь на размере бина. Чем больше наблюдений в бине, тем больший процент данных можно взять для тестовой выборки.

python

Копіювати код

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Пример данных

data = pd.DataFrame({

'Ages': [20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90],

'Feature1': range(15),

'Feature2': range(15, 30)

})

# Разбиваем на возрастные бины

bins = [20, 30, 40, 50, 60, 70, 80, 90]

labels = ['20-30', '30-40', '40-50', '50-60', '60-70', '70-80', '80-90']

data['AgeBin'] = pd.cut(data['Ages'], bins=bins, labels=labels)

# Пустые DataFrame'ы для тренировочной и тестовой выборки

train\_data = pd.DataFrame()

test\_data = pd.DataFrame()

# Пропорциональное разбиение для каждого бина

for bin\_label in labels:

bin\_data = data[data['AgeBin'] == bin\_label]

# Пропорциональный размер тестового набора зависит от количества данных в бине

if len(bin\_data) > 1: # Проверяем, что есть больше одного элемента для разделения

test\_size = min(0.3, 1 / len(bin\_data)) # Чем меньше данных, тем меньший тестовый набор

train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=test\_size, random\_state=42)

train\_data = pd.concat([train\_data, train\_bin], axis=0)

test\_data = pd.concat([test\_data, test\_bin], axis=0)

# Сбрасываем индексы

train\_data = train\_data.reset\_index(drop=True)

test\_data = test\_data.reset\_index(drop=True)

print("Training data:")

print(train\_data)

print("\nTest data:")

print(test\_data)

**Пример 2: Минимальная доля данных для каждого бина**

Если некоторые возрастные категории содержат очень мало данных (например, всего 1 или 2 наблюдения), можно задать минимальную долю данных для тестового набора.

python

Копіювати код

for bin\_label in labels:

bin\_data = data[data['AgeBin'] == bin\_label]

if len(bin\_data) > 1:

# Минимальная доля данных для каждого бина (например, не менее 1 записи в тестовом наборе)

test\_size = max(0.2, 1 / len(bin\_data)) # Минимум одна запись в тесте, но не более 20%

train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=test\_size, random\_state=42)

train\_data = pd.concat([train\_data, train\_bin], axis=0)

test\_data = pd.concat([test\_data, test\_bin], axis=0)

**Пример 3: Использование стратификации**

Если ваши данные сильно разбалансированы, и вам нужно сохранить пропорции возрастных групп в тренировочном и тестовом наборах, лучше всего использовать параметр **стратификации** в train\_test\_split, что позволит автоматически сохранить пропорции возрастов в обеих выборках.

python

Копіювати код

# Без разбиения на бины вручную, просто используем стратификацию

X = data.drop(['Ages', 'AgeBin'], axis=1) # Признаки

y = data['Ages'] # Целевая переменная (возраст)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=data['AgeBin'], random\_state=42)

print("Training set:")

print(X\_train)

print("\nTest set:")

print(X\_test)

**Итоги:**

* **Пропорциональное разбиение** позволяет учесть разные размеры бинов и адаптировать процент данных для тестовой выборки.
* **Минимальная доля данных** для каждого бина гарантирует, что даже маленькие группы попадут в тестовую выборку.
* **Стратификация** — это удобный способ сохранить пропорции возрастных групп в выборках без явного разбиения на бины.

Выбор подхода зависит от того, как сильно различаются размеры возрастных групп в данных и насколько критично сохранить их пропорции.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

Как работают методы стратификации?

**Програма ChatGPT сказала:**



**Стратификация** — это метод, который используется для разбиения выборки таким образом, чтобы пропорции целевой переменной (классов или категорий) оставались примерно одинаковыми как в обучающей, так и в тестовой выборках. Этот подход особенно полезен в случаях, когда данные неравномерно распределены между классами (например, есть дисбаланс классов) или когда нужно сохранить пропорции возрастных или других категорий при разбиении.

**Как работает стратификация:**

1. **Выбор целевой переменной для стратификации**: Стратификация производится по целевой переменной, то есть по переменной, значения которой должны быть равномерно распределены между обучающей и тестовой выборками. Например, если целевая переменная — это возрастная категория или класс (например, больные и здоровые пациенты), стратификация гарантирует, что каждый класс будет представлен в обеих выборках в той же пропорции, что и в исходных данных.
2. **Процесс разбиения**:
   * **Пропорции классов сохраняются**: В процессе разбиения данные сортируются по выбранной целевой переменной (например, возрастной группе или классу). Затем данные для каждого класса делятся на тренировочную и тестовую выборки таким образом, чтобы пропорции остались как можно ближе к исходным.
   * Если, например, класс "A" занимает 30% от всей выборки, а класс "B" — 70%, то при разбиении выборки с помощью стратификации эти пропорции сохранятся как в обучающем, так и в тестовом наборах данных.

**Пример с использованием функции train\_test\_split в Python**

Библиотека scikit-learn предоставляет удобную функцию train\_test\_split, которая поддерживает стратификацию.

**Пример на Python:**

python

Копіювати код

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Пример данных

data = pd.DataFrame({

'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1],

'Target': ['A', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A'] # Целевая переменная с классами

})

# Признаки

X = data[['Feature1', 'Feature2']]

# Целевая переменная

y = data['Target']

# Разбиение с использованием стратификации

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=42)

# Результаты

print("Training set class distribution:")

print(y\_train.value\_counts())

print("\nTest set class distribution:")

print(y\_test.value\_counts())

**Объяснение:**

* В этом примере используется целевая переменная y, которая содержит два класса: "A" и "B".
* Мы делим данные на тренировочную и тестовую выборки с использованием параметра stratify=y, что гарантирует сохранение пропорций классов "A" и "B" в обоих наборах.
* Параметр test\_size=0.3 означает, что 30% данных будут использоваться для тестовой выборки, а 70% — для тренировочной.

**Когда использовать стратификацию:**

1. **Дисбаланс классов**: Если в выборке наблюдается дисбаланс классов (например, один класс доминирует), стратификация поможет избежать проблемы, когда один из классов практически не будет представлен в тестовой выборке.
2. **Гарантия равномерного представления категорий**: Если целевая переменная представляет собой категорию (например, возрастные группы, медицинские состояния), стратификация помогает гарантировать, что каждая категория будет представлена в обеих выборках.
3. **Стратификация для регрессии**: Хотя чаще всего стратификация используется для задач классификации, её можно применять и в задачах регрессии. Например, если вы работаете с непрерывной переменной (например, возраст), можно предварительно разделить её на интервалы (бины) и провести стратификацию по этим интервалам.

**Пример стратификации по возрастным группам (регрессия):**

python

Копіювати код

# Пример данных с возрастом (регрессионная задача)

data = pd.DataFrame({

'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1],

'Age': [18, 22, 25, 35, 45, 55, 65, 75, 85, 95] # Возраст (непрерывная переменная)

})

# Признаки

X = data[['Feature1', 'Feature2']]

# Бины для возрастных групп (например, интервалы по 20 лет)

data['AgeGroup'] = pd.cut(data['Age'], bins=[0, 20, 40, 60, 80, 100], labels=['0-20', '20-40', '40-60', '60-80', '80-100'])

# Разбиение данных с использованием стратификации по возрастным группам

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, data['Age'], test\_size=0.3, stratify=data['AgeGroup'], random\_state=42)

# Проверка распределения возрастных групп

print("Training set age group distribution:")

print(pd.cut(y\_train, bins=[0, 20, 40, 60, 80, 100]).value\_counts())

print("\nTest set age group distribution:")

print(pd.cut(y\_test, bins=[0, 20, 40, 60, 80, 100]).value\_counts())

**Плюсы стратификации:**

* **Баланс**: Обеспечивает, что целевая переменная будет представлена в обеих выборках примерно одинаково.
* **Предотвращает потерю данных**: Важные классы или категории не будут пропущены в тестовой выборке, что делает оценку модели более точной.
* **Репрезентативность**: Тестовая выборка будет более репрезентативной и ближе по распределению к общей выборке.

**Минусы стратификации:**

* Если классы слишком малочисленны (например, один из классов содержит только одну запись), это может затруднить стратифицированное разбиение, поскольку каждая группа должна содержать хотя бы две записи.

**Вывод:**

Стратификация — это мощный инструмент для правильного разбиения данных, особенно при наличии дисбаланса классов или категорий. Она помогает сохранить пропорции целевой переменной в обеих выборках, что делает модель более устойчивой и позволяет корректно оценить её точность. В задачах классификации стратификация позволяет избежать доминирования одного класса, а в регрессии — равномерно распределить данные по группам.

[Джерело](https://gptonline.ai/ru/)

**Ви сказали:**

В предыдущем запросе к тебе test\_size = max(0.2, 1 / len(bin\_data)) # Минимум одна запись в тесте, но не более 20%, тут точно так?

**Програма ChatGPT сказала:**



В приведённом фрагменте кода, где используется строка test\_size = max(0.2, 1 / len(bin\_data)), цель заключается в следующем:

* **0.2**: Это фиксированное значение, означающее, что минимум 20% данных будут выделены для тестовой выборки.
* **1 / len(bin\_data)**: Этот термин гарантирует, что если данных в бине очень мало, хотя бы одна запись попадёт в тестовую выборку. Если в бине, например, всего 3 записи, то значение 1 / 3 ≈ 0.33, и в тестовую выборку попадёт одна запись.

Однако **логика в коде не совсем корректна** и может привести к нежелательным результатам, когда размер бина очень мал. Давайте разберём почему:

**Проблемы:**

1. **При малом размере бина**: Например, если в бине всего 2 записи, то test\_size = max(0.2, 1 / 2) = 0.5. Это значит, что 50% данных пойдут в тест, что может быть слишком большим процентом при очень малых объёмах данных.
2. **При большом размере бина**: Например, если в бине 10 записей, то 1 / 10 = 0.1, но max(0.2, 0.1) вернёт 0.2, и это может быть достаточно, но не оптимально для бинов с большим количеством данных.

**Как улучшить:**

Для малых выборок лучше не использовать фиксированное значение в виде 0.2, а подбирать размер тестовой выборки более гибко. Один из подходов — это установить, что при малых объёмах данных будет выбираться минимум одна запись в тестовый набор, но с ограничением максимального процента.

**Правильный вариант:**

python

Копіювати код

# Устанавливаем минимум 1 запись в тестовой выборке, но не более 20% данных

test\_size = min(0.2, max(1 / len(bin\_data), 0.2))

**Объяснение:**

* **max(1 / len(bin\_data), 0.2)**: Этот кусок гарантирует, что если данных мало (например, всего 1 или 2 записи), хотя бы одна запись будет выделена в тестовую выборку. Но если данных достаточно, то минимум 20% (0.2) будут отобраны для теста.
* **min(0.2, ...)**: Это ограничивает процент тестовой выборки сверху, чтобы в случае больших выборок не превышать 20%.

**Пример:**

* Если в бине **3 записи**: 1 / 3 = 0.33, тогда тестовая выборка будет содержать **1 запись**.
* Если в бине **20 записей**: 1 / 20 = 0.05, но будет выбрано минимум **20%** (то есть **4 записи**).

**Полный пример кода:**

python

Копіювати код

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Пример данных

data = pd.DataFrame({

'Ages': [20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80],

'Feature1': range(13),

'Feature2': range(13, 26)

})

# Разбиваем на возрастные бины

bins = [20, 30, 40, 50, 60, 70, 80]

labels = ['20-30', '30-40', '40-50', '50-60', '60-70', '70-80']

data['AgeBin'] = pd.cut(data['Ages'], bins=bins, labels=labels)

# Пустые DataFrame'ы для тренировочной и тестовой выборки

train\_data = pd.DataFrame()

test\_data = pd.DataFrame()

# Пропорциональное разбиение для каждого бина

for bin\_label in labels:

bin\_data = data[data['AgeBin'] == bin\_label]

if len(bin\_data) > 1: # Проверяем, что есть больше одного элемента для разделения

test\_size = min(0.2, max(1 / len(bin\_data), 0.2)) # Оптимизация размера тестовой выборки

train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=test\_size, random\_state=42)

train\_data = pd.concat([train\_data, train\_bin], axis=0)

test\_data = pd.concat([test\_data, test\_bin], axis=0)

# Сбрасываем индексы

train\_data = train\_data.reset\_index(drop=True)

test\_data = test\_data.reset\_index(drop=True)

print("Training data:")

print(train\_data)

print("\nTest data:")

print(test\_data)

**Вывод:**

Использование функции max(1 / len(bin\_data), 0.2) для расчёта размера тестовой выборки в некоторых случаях не работает корректно, когда размер бина слишком мал или слишком велик. Оптимизировав разбиение через min(0.2, max(1 / len(bin\_data), 0.2)), вы гарантируете, что хотя бы одна запись попадёт в тестовую выборку, при этом сохраняя разумный процент данных для каждой возрастной категории.

**Ви сказали:**

for bin\_label in labels: bin\_data = data[data['AgeBin'] == bin\_label] if len(bin\_data) > 1: # Минимальная доля данных для каждого бина (например, не менее 1 записи в тестовом наборе) test\_size = max(0.2, 1 / len(bin\_data)) # Минимум одна запись в тесте, но не более 20% train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=test\_size, random\_state=42) train\_data = pd.concat([train\_data, train\_bin], axis=0) test\_data = pd.concat([test\_data, test\_bin], axis=0) А сдесь не min вместо max стоит?

**Програма ChatGPT сказала:**



Да, вы абсолютно правы! В вашем коде действительно нужно использовать **min вместо max**.

Поясню:

* **test\_size = max(0.2, 1 / len(bin\_data))** в текущем виде делает так, что всегда выбирается **наибольшее** значение между фиксированными 20% и долей, равной 1 записи на количество элементов в бине. Это означает, что в случае малых выборок (например, 2-3 записи), может быть выбрано слишком много данных для теста.

Например:

* + Если в бине 3 записи, то 1 / 3 = 0.33. В таком случае max(0.2, 0.33) выберет 33%, что может быть слишком много для небольших бинов.

**Нужно использовать min(0.2, 1 / len(bin\_data)), чтобы:**

* **Максимальный размер тестовой выборки не превышал 20%** от данных в бине, если данных много.
* **Минимально выделять хотя бы одну запись** в тестовую выборку при малом количестве данных в бине (когда количество записей в бине мало).

**Правильный вариант:**

python

Копіювати код

for bin\_label in labels:

bin\_data = data[data['AgeBin'] == bin\_label]

if len(bin\_data) > 1:

# Минимальная доля данных для каждого бина (например, не менее 1 записи в тестовом наборе)

test\_size = min(0.2, 1 / len(bin\_data)) # Максимум 20%, но минимум 1 запись в тесте

train\_bin, test\_bin = train\_test\_split(bin\_data, test\_size=test\_size, random\_state=42)

train\_data = pd.concat([train\_data, train\_bin], axis=0)

test\_data = pd.concat([test\_data, test\_bin], axis=0)

**Объяснение:**

* **min(0.2, 1 / len(bin\_data))** означает:
  + Если данных много (например, больше 5 в бине), то для теста будет выделено не более 20%.
  + Если данных мало (например, 2 или 3 записи), то тестовая выборка всё равно получит хотя бы одну запись.

Этот подход обеспечивает более гибкое и корректное разбиение данных на тренировочную и тестовую выборки, особенно когда в разных бинах количество данных значительно отличается.

**Ви сказали:**

Как применять stratify в X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42) ?

**Програма ChatGPT сказала:**



Чтобы правильно использовать **стратификацию** в функции train\_test\_split, нужно передать параметр **stratify=y**. Это гарантирует, что пропорции классов или категорий целевой переменной yyy сохранятся как в тренировочной, так и в тестовой выборках.

**Синтаксис с stratify:**

python

Копіювати код

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.1, random\_state=42, stratify=y

)

**Как это работает:**

* **stratify=y**: Делает так, чтобы пропорции классов в переменной yyy (целевой переменной) были одинаковыми и в обучающей, и в тестовой выборках.
* Если вы работаете, например, с задачей классификации, где классы представлены неравномерно, использование stratify помогает избежать дисбаланса в тренировочном и тестовом наборах.

Пример:

* Если в данных есть два класса:
  + Класс 0: 90% записей
  + Класс 1: 10% записей  
    Тогда и в тренировочной, и в тестовой выборках эти пропорции будут сохранены.

**Пример на Python:**

python

Копіювати код

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Создаём пример данных с неравномерными классами

data = pd.DataFrame({

'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1],

'Target': [0, 0, 0, 0, 1, 0, 0, 1, 1, 1] # Классы 0 и 1

})

# Разделяем признаки и целевую переменную

X = data[['Feature1', 'Feature2']]

y = data['Target']

# Выполняем разбиение с использованием стратификации

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42, stratify=y

)

# Проверяем распределение классов в тренировочной и тестовой выборках

print("Распределение классов в тренировочной выборке:")

print(y\_train.value\_counts())

print("\nРаспределение классов в тестовой выборке:")

print(y\_test.value\_counts())

**Объяснение:**

* В этом примере пропорции классов **0** и **1** в исходных данных сохраняются и в тренировочном, и в тестовом наборе.
* Стратификация гарантирует, что модель будет иметь репрезентативные данные для обучения и тестирования, даже при наличии классов с малой численностью.

**Ошибки, которых нужно избегать:**

* **Если один из классов встречается только один раз**, использование стратификации приведёт к ошибке:

kotlin

Копіювати код

ValueError: The least populated class in y has only 1 member, which is too few.

Чтобы это исправить, можно либо удалить редкие классы, либо объединить их с другими категориями, либо применить случайное разбиение без стратификации.

Использование stratify — это хорошая практика при работе с дисбалансированными данными, чтобы избежать смещения модели в сторону более представленных классов.

**Ви сказали:**

Корреляция пирсона - это линейная корреляция?

**Програма ChatGPT сказала:**



Да, **корреляция Пирсона** измеряет **линейную зависимость** между двумя переменными. Она показывает, насколько хорошо изменения одной переменной предсказывают изменения другой при условии, что эта зависимость **линейная**. Этот коэффициент корреляции принимает значения от **-1 до 1**:

* **1**: Полная положительная линейная зависимость (при увеличении одной переменной другая также увеличивается).
* **0**: Отсутствие линейной зависимости.
* **-1**: Полная отрицательная линейная зависимость (при увеличении одной переменной другая уменьшается).

**Формула корреляции Пирсона:**

r=∑(xi−xˉ)(yi−yˉ)∑(xi−xˉ)2∑(yi−yˉ)2r = \frac{\sum{(x\_i - \bar{x})(y\_i - \bar{y})}}{\sqrt{\sum{(x\_i - \bar{x})^2} \sum{(y\_i - \bar{y})^2}}}r=∑(xi​−xˉ)2∑(yi​−yˉ​)2​∑(xi​−xˉ)(yi​−yˉ​)​

Где:

* rrr — коэффициент корреляции Пирсона,
* xix\_ixi​ и yiy\_iyi​ — значения переменных,
* xˉ\bar{x}xˉ и yˉ\bar{y}yˉ​ — средние значения переменных.

**Условия применения:**

* Корреляция Пирсона применима только для **линейных зависимостей**. Если зависимость между переменными нелинейная, этот коэффициент может быть близок к нулю, даже если переменные явно связаны.
* Данные должны быть **количественными** и **нормально распределёнными** (или близки к нормальному распределению).
* Должны отсутствовать значительные выбросы, так как они могут сильно исказить результат.

**Пример на Python:**

python

Копіювати код

import pandas as pd

import numpy as np

# Пример данных

data = pd.DataFrame({

'X': [1, 2, 3, 4, 5],

'Y': [2, 4, 6, 8, 10] # Линейная зависимость Y = 2\*X

})

# Вычисляем корреляцию Пирсона

correlation = data['X'].corr(data['Y'], method='pearson')

print(f"Коэффициент корреляции Пирсона: {correlation}")

**Результат**: В этом примере корреляция Пирсона будет равна **1**, так как между переменными X и Y существует идеальная положительная линейная зависимость.

**Примечание:**

Если зависимость **нелинейная** (например, квадратичная), то коэффициент Пирсона может быть близок к нулю, даже если связь между переменными существует. В таких случаях лучше использовать другие меры, такие как **корреляция Спирмена** или **Кендалла**, которые работают с ранговыми данными и могут выявлять монотонные зависимости.

Корреляция Пирсона — отличный инструмент для оценки **линейных** взаимосвязей, но для более сложных зависимостей нужны другие методы.

[Джерело](https://gptonline.ai/ru/)

Початок форми

Кінець форми

Доступна нова версія GPT. Продовжуйте спілкуватися, користуючись старою версією, або почніть [новий чат](/g/g-mzFm1dKjW-chat-gpt), щоб використовувати останню версію.