**Неперервні числові дані** можуть бути перетворені за допомогою різних стратегій. Нижче наведено порівняння між різними стратегіями перетворення ознак:

1. **Квантування (Binning)**: Розбиття неперервних числових даних на інтервали (bins) для створення категорій або числових діапазонів. Наприклад, якщо ми маємо набір даних з віком людей, ми можемо розділити їх на категорії, такі як “діти”, “підлітки”, “дорослі” та “літні люди”. Використовуючи бібліотеку Pandas, ми можемо використовувати функцію cut() для квантування даних. Наприклад, наступний код розділить вік на 4 категорії: “діти”, “підлітки”, “дорослі” та “літні люди”.

import pandas as pd

# Create a DataFrame with ages

df = pd.DataFrame({'age': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]})

# Define the bins

bins = [0, 18, 35, 60, 100]

# Define the labels

labels = ['діти', 'підлітки', 'дорослі', 'літні люди']

# Cut the data into bins

df['age\_group'] = pd.cut(df['age'], bins=bins, labels=labels)

# Print the result

print(df)

1. **Бінаризація (Binarization)**: Перетворення неперервних числових ознак у бінарні за допомогою порогового значення. Наприклад, якщо ми маємо набір даних з віком людей, ми можемо перетворити його на бінарний набір даних, де 1 означає, що людина молодша за 30 років, а 0 - старше 30 років. Використовуючи бібліотеку NumPy, ми можемо використовувати функцію where() для бінаризації даних. Наприклад, наступний код перетворить вік на бінарний набір даних:

import numpy as np

# Create an array with ages

ages = np.array([10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80])

# Binarize the data

binarized\_ages = np.where(ages < 30, 1, 0)

# Print the result

print(binarized\_ages)

1. **Адаптивний бінінг (Adaptive Binning)**: Автоматичне розділення числових значень в інтервали на основі їх розподілу. Наприклад, якщо ми маємо набір даних з віком людей, ми можемо автоматично розділити його на категорії на основі розподілу віку. Використовуючи бібліотеку Python, таку як vorbin, ми можемо використовувати алгоритм Вороного для автоматичного розділення даних на кластери з однаковим сигналом-шумом. Наприклад, наступний код демонструє використання бібліотеки vorbin для адаптивного бінінгу даних:

import numpy as np

from vorbin.voronoi\_2d\_binning import voronoi\_2d\_binning

# Create an array with data

data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])

# Define the signal-to-noise ratio

target\_sn = 10.0

# Perform adaptive binning

bin\_number, x\_gen, y\_gen, x\_bar, y\_bar, sn, nPixels, scale = voronoi\_2d\_binning(data[:, 0], data[:, 1], np.ones(len(data)), np.ones(len(data)), target\_sn)

# Print the result

print(bin\_number)

У цьому прикладі ми створюємо набір даних з двох стовпців, використовуючи бібліотеку NumPy. Потім ми використовуємо функцію voronoi\_2d\_binning() з бібліотеки vorbin, щоб автоматично розділити дані на кластери з однаковим сигналом-шумом. Результатом є масив, який містить номер кластера для кожного рядка даних.

1. **Reframe Numerical Quantities** - це зміна шкали або одиниць вимірювання числових даних для спрощення розуміння. Наприклад, якщо ми маємо набір даних з вагою людей у фунтах, ми можемо перевести його в кілограми для спрощення розуміння. Використовуючи бібліотеку NumPy, ми можемо використовувати функцію multiply() для зміни шкали даних. Наприклад, наступний код переведе вагу з фунтів в кілограми:

import numpy as np

# Create an array with weights in pounds

weights\_pounds = np.array([100, 150, 200, 250, 300])

# Convert the weights to kilograms

weights\_kilograms = np.multiply(weights\_pounds, 0.453592)

# Print the result

print(weights\_kilograms)

1. **Рангова трансформація (Rank Transformation)** - це заміна числових значень їх рангами або позиціями у впорядкованому рядку. Це може бути корисно для зменшення впливу викидів або для зменшення впливу на розподіл даних. Використовуючи бібліотеку SciPy, ми можемо використовувати функцію rankdata() для рангової трансформації даних. Наприклад, наступний код замінить числові значення їх рангами:

import numpy as np

from scipy.stats import rankdata

data = np.array([10, 20, 30, 40, 50])

ranked\_data = rankdata(data)

print(ranked\_data)

# output: [1. 2. 3. 4. 5.]

У цьому прикладі ми створюємо набір даних з одного стовпця, використовуючи бібліотеку NumPy. Потім ми використовуємо функцію rankdata() з бібліотеки SciPy, щоб замінити числові значення їх рангами. Результатом є масив, який містить ранг для кожного рядка даних.

1. **Перетворення, що стабілізують дисперсію (Variance-Stabilizing Transformations)** - це методи зменшення дисперсії в числових даних. Ці методи можуть бути корисними для зменшення впливу викидів або для зменшення впливу на розподіл даних. Наприклад, якщо ми маємо набір даних з висотою людей, ми можемо використовувати перетворення Бокса-Кокса для стабілізації дисперсії даних. Використовуючи бібліотеку SciPy, ми можемо використовувати функцію boxcox() для перетворення даних. Наприклад, наступний код застосовує перетворення Бокса-Кокса до висоти людей:

import numpy as np

from scipy.stats import boxcox

# Create an array with heights

heights = np.array([150, 160, 170, 180, 190])

# Apply the Box-Cox transformation

transformed\_heights, lambda\_value = boxcox(heights)

# Print the result

print(transformed\_heights)

# [47.54835977 49.83582907 52.08182972 54.28949129 56.46154736]

1. **Боротьба з викидами (Outlier Handling)** - це техніки, такі як трімінг (вилучення) або віндсоризація (заміна викидів). Ці техніки можуть бути корисними для зменшення впливу викидів на аналіз даних. Наприклад, якщо ми маємо набір даних з висотою людей, ми можемо використовувати трімінг для вилучення викидів з даних. Використовуючи бібліотеку NumPy, ми можемо використовувати функцію percentile() для визначення границь викидів. Наприклад, наступний код вилучить викиди з даних про висоту людей:

import numpy as np

# Create an array with heights

heights = np.array([150, 160, 170, 180, 190, 200, 210, 220, 230, 240])

# Calculate the 25th and 75th percentiles

q25, q75 = np.percentile(heights, [25, 75])

# Calculate the interquartile range

iqr = q75 - q25

# Calculate the lower and upper bounds for outliers

lower\_bound = q25 - (1.5 \* iqr)

upper\_bound = q75 + (1.5 \* iqr)

# Trim the data

trimmed\_heights = heights[(heights >= lower\_bound) & (heights <= upper\_bound)]

# Print the result

print(trimmed\_heights)

У цьому прикладі ми створюємо набір даних з висотою людей, використовуючи бібліотеку NumPy. Потім ми використовуємо функцію percentile() для визначення границь викидів. Наступні дві стрічки коду визначають нижню та верхню границі викидів, а остання стрічка коду вилучає викиди з даних. Результатом є масив, який м

1. **Шкалювання або нормалізація (Scaling or Normalization)** - це процес приведення числових даних до певного діапазону, наприклад, шкалювання від 0 до 1 або нормалізація на основі середнього та стандартного відхилення. Це може бути корисно для зменшення впливу великих числових значень на аналіз даних. Використовуючи бібліотеку Scikit-learn, ми можемо використовувати функцію MinMaxScaler() для шкалювання даних до діапазону від 0 до 1. Наприклад, наступний код шкалює дані з висотою людей:

import numpy as np

from sklearn.preprocessing import MinMaxScaler

# Create an array with heights

heights = np.array([150, 160, 170, 180, 190])

# Scale the data

scaler = MinMaxScaler()

scaled\_heights = scaler.fit\_transform(heights.reshape(-1, 1))

# Print the result

print(scaled\_heights)

У цьому прикладі ми створюємо набір даних з висотою людей, використовуючи бібліотеку NumPy. Потім ми використовуємо функцію MinMaxScaler() з бібліотеки Scikit-learn, щоб шкалювати дані до діапазону від 0 до 1. Результатом є масив, який містить шкальовані значення висоти.

Для кодування **категоріальних даних** існує кілька стратегій, кожна з яких має свої переваги та недоліки. Ось порівняння чотирьох таких стратегій:

1. **Дамі-кодування (Dummy Encoding)**: Створення фіктивних (дамі) змінних для представлення категоріальних даних. Кожна унікальна категорія перетворюється на окрему змінну, яка приймає значення 0 або 1, в залежності від того, чи належить кожен запис до цієї категорії. Цей метод підходить для категоріальних даних з невеликою кількістю унікальних значень. Ось приклад кодування категоріальних даних за допомогою бібліотеки Pandas:

import pandas as pd

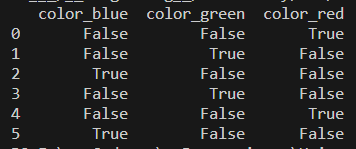
# Створення датафрейму з даними

df = pd.DataFrame({'color': ['red', 'green', 'blue', 'green', 'red', 'blue']})

# Використання методу get\_dummies для кодування категорій

df\_encoded = pd.get\_dummies(df, columns=['color'])

print(df\_encoded)



У цьому прикладі ми створюємо датафрейм з даними, які містять категоріальну ознаку “color”. Ми використовуємо метод get\_dummies для кодування категорій. Цей метод перетворює категоріальну ознаку у вектор, що складається з нулів та однієї одиниці. Кожна категорія закодована у вигляді окремого стовпця. Результат виводиться на екран.

1. **Label Encoding**: Призначення числових міток категоріям або рівням категорій для кодування категоріальних ознак. Кожна унікальна категорія отримує унікальне числове значення. Цей метод підходить для категоріальних даних з великою кількістю унікальних значень. Ось приклад кодування категоріальних даних за допомогою бібліотеки Scikit-learn:

from sklearn.preprocessing import LabelEncoder

import pandas as pd

# Створення датафрейму з даними

df = pd.DataFrame({'color': ['red', 'green', 'blue', 'green', 'red', 'blue']})

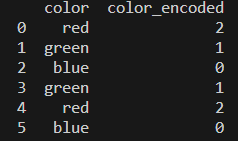
# Створення екземпляру LabelEncoder

le = LabelEncoder()

# Кодування категорій

df['color\_encoded'] = le.fit\_transform(df['color'])

print(df)



У цьому прикладі ми створюємо датафрейм з даними, які містять категоріальну ознаку “color”. Ми створюємо екземпляр LabelEncoder та використовуємо його для кодування категорій. Кожна унікальна категорія отримує унікальне числове значення. Результат виводиться на екран.

1. **One-Hot Encoding**: Перетворення категоріальних даних у вектори бінарних значень для кожної унікальної категорії. Цей метод підходить для категоріальних даних з великою кількістю унікальних значень. Ось приклад кодування категоріальних даних за допомогою бібліотеки Scikit-learn:

from sklearn.preprocessing import OneHotEncoder

import pandas as pd

# Створення датафрейму з даними

df = pd.DataFrame({'color': ['red', 'green', 'blue', 'green', 'red', 'blue']})

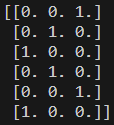
# Створення екземпляру OneHotEncoder

ohe = OneHotEncoder()

# Кодування категорій

df\_encoded = ohe.fit\_transform(df[['color']])

print(df\_encoded.toarray())



У цьому прикладі ми створюємо датафрейм з даними, які містять категоріальну ознаку “color”. Ми створюємо екземпляр OneHotEncoder та використовуємо його для кодування категорій. Кожна унікальна категорія перетворюється на вектор, що складається з нулів та однієї одиниці. Результат виводиться на екран.

1. **Feature Hashing**: Використання хеш-функцій для перетворення категоріальних ознак у вектори фіксованої довжини. Цей метод підходить для категоріальних даних з великою кількістю унікальних значень. Ось приклад кодування категоріальних даних за допомогою бібліотеки Scikit-learn:

from sklearn.feature\_extraction import FeatureHasher

import pandas as pd

df = pd.DataFrame({'color': [("red", 'green', 'blue'), ('green','green'), 'blue', 'green', 'red', 'blue']})

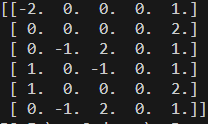
# Створення екземпляру FeatureHasher

fh = FeatureHasher(n\_features=5, input\_type='string')

# Кодування категорій

df\_encoded = fh.transform(df['color'])

print(df\_encoded.toarray())



У цьому прикладі ми створюємо датафрейм з даними, які містять категоріальну ознаку “color”. Ми створюємо екземпляр FeatureHasher та використовуємо його для кодування категорій. Кожна унікальна категорія перетворюється на вектор фіксованої довжини. Результат виводиться на екран.