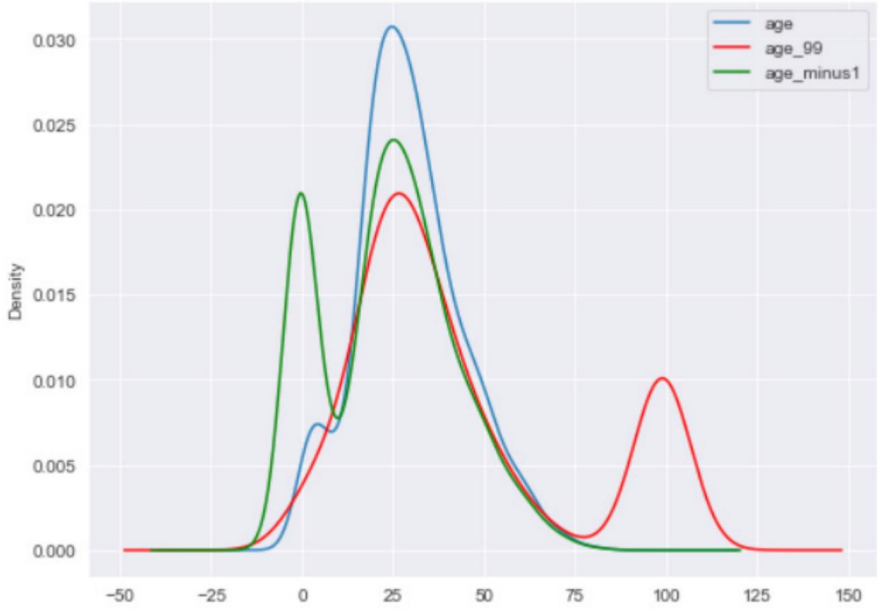
Bước cuối cùng là vẽ biểu đồ mật độ hạt nhân cho cột tuổi ban đầu và cho các cột tuổi mà các giá trị bị thiếu được thay thế bằng 99 và 1. Đoạn mã sau thực hiện điều đó:

Kịch bản 15:

```
plt.rcParams["figure.figsize"] = [8,6]

fig = plt.figure()
ax = fig.add_subplot(111)

titanic_data['tuổi'] .plot(loại='kde', ax=ax)

titanic_data['age_99'] .plot(loại='kde', ax=ax, màu='đỏ')

titanic_data['age_minus1'] .plot(loại='kde', ax=ax, màu='xanh
lá cây')

dòng, nhãn = ax.get_legend_handles_labels() ax.legend(dòng,
nhãn, loc='tốt nhất')
```

Đầu ra:



Ưu điểm và nhược điểm của việc quy ước giá trị tùy ý tương tự như việc quy ước cuối phân phối.

Điều quan trọng cần đề cập là việc quy giá trị tùy ý cũng có thể được sử dụng cho dữ liệu phân loại. Trong trường hợp dữ liệu phân loại, bạn chỉ cần thêm giá trị missing vào các cột thiếu giá trị phân loại.

Trong phần này, chúng ta đã nghiên cứu ba cách tiếp cận để xử lý dữ liệu số bị thiếu. Trong phần tiếp theo, bạn sẽ thấy cách xử lý dữ liệu danh mục bị thiếu.

3.4 Xử lý dữ liệu danh mục bị thiếu

3.4.1. Quy kết danh mục thường xuyên

Một trong những cách phổ biến nhất để xử lý các giá trị bị thiếu trong cột phân loại là thay thế các giá trị bị thiếu bằng các giá trị xuất hiện thường xuyên nhất, tức là chế độ của cột.
Vì lý do này, phép quy ước thể loại thường xuyên cũng được gọi là phép quy ước chế độ. Chúng ta hãy xem một ví dụ thực tế về phép quy ước thể loại thường xuyên.

Chúng tôi sẽ lại sử dụng tập dữ liệu Titanic. Đầu tiên chúng tôi sẽ cố gắng tìm phần trăm các giá trị bị thiếu trong độ tuổi, giá vé và số người lên tàu cột thị trấn .

Kịch bản 16:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns

plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

titanic_data = sns.load_dataset('titanic')

titanic_data = titanic_data[["embark_town", "tuổi", "giá vé"]]
titanic_data.đầu()
titanic_data.isnull().mean()
```

Đầu ra:

```
thị trấn_đi_vào 0.002245
tuổi 0,198653
giá vé 0.000000
Kiểu dữ liệu: float64
```

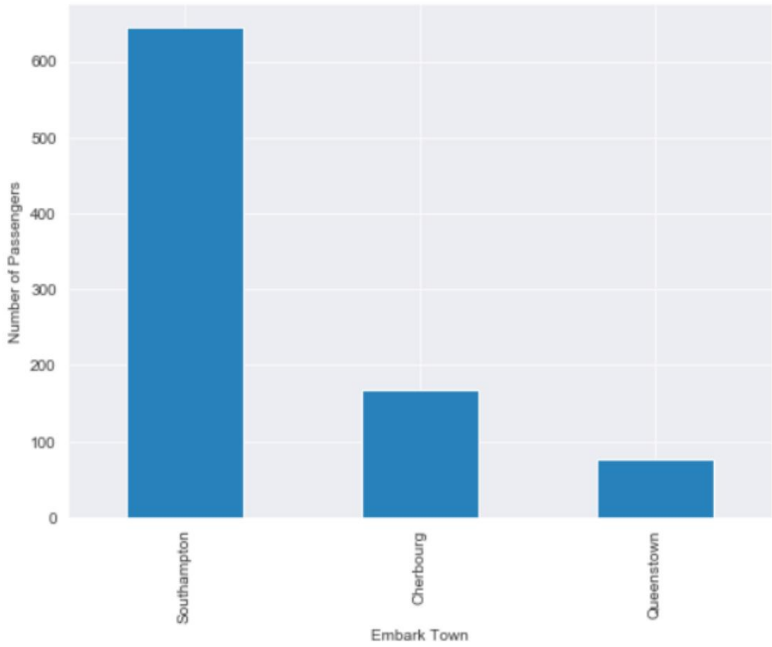
Đầu ra cho thấy các cột embark_town và age có giá trị bị thiếu. Tỷ lệ giá trị bị thiếu cho embark_
cột town rất ít. Hãy vẽ biểu đồ thanh hiển thị từng danh mục trong cột embark_town so với số lượng hành khách.

Kịch bản 17:

```
titanic_data.embark_town.value_counts().sort_
giá trị(tăng dần=Sai).plot.bar()
plt.xlabel('Vào thị trấn')
plt.ylabel('Số lượng hành khách')
```

Dữ liệu đầu ra cho thấy rõ ràng rằng hầu hết hành khách lên tàu đều xuất phát từ Southampton.

Đầu ra:



Hãy cùng kiểm tra xem Southampton có thực sự là giá trị chế độ cho cột embark_town hay không .

Kịch bản 18:

```
titanic_data.embark_town.mode()
```

Đầu ra:

```
0 Nam Định
dtype: đối tượng
```

Tiếp theo, chúng ta có thể chỉ cần thay thế các giá trị bị thiếu trong cột thị trấn adopt bằng Southampton.

Kịch bản 19:

```
titanic_data.embark_town.fillna('Southampton', inplace=Đúng)
```

Bây giờ chúng ta hãy tìm mốt của cột tuổi và sử dụng nó để thay thế các giá trị bị thiếu trong cột tuổi .

Kịch bản 20:

```
titanic_data.age.mode()
```

Đầu ra:

24.0

Đầu ra cho thấy mốt của cột tuổi là 24.
Vì vậy, chúng ta có thể sử dụng giá trị này để thay thế các giá trị bị thiếu trong cột tuổi .

Kịch bản 21:

```
nhập numpy dưới dạng np


titanic_data['age_mode'] = titanic_data.age.fillna(24)


titanic_data.head(20)
```

Đầu ra:

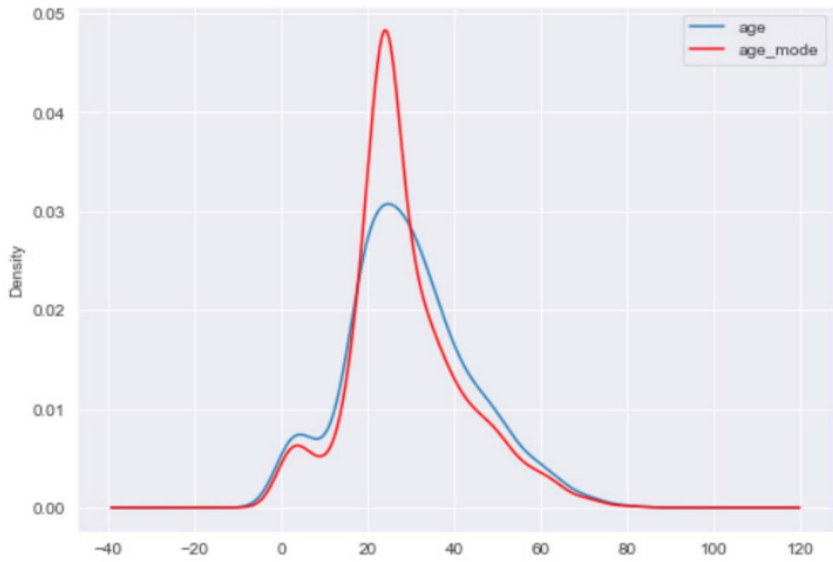
	embark_town	age	fare	age_mode
0	Southampton	22.0	7.2500	22.0
1	Cherbourg	38.0	71.2833	38.0
2	Southampton	26.0	7.9250	26.0
3	Southampton	35.0	53.1000	35.0
4	Southampton	35.0	8.0500	35.0
5	Queenstown	NaN	8.4583	24.0
6	Southampton	54.0	51.8625	54.0
7	Southampton	2.0	21.0750	2.0
8	Southampton	27.0	11.1333	27.0
9	Cherbourg	14.0	30.0708	14.0
10	Southampton	4.0	16.7000	4.0
11	Southampton	58.0	26.5500	58.0
12	Southampton	20.0	8.0500	20.0
13	Southampton	39.0	31.2750	39.0
14	Southampton	14.0	7.8542	14.0
15	Southampton	55.0	16.0000	55.0
16	Queenstown	2.0	29.1250	2.0
17	Southampton	NaN	13.0000	24.0
18	Southampton	31.0	18.0000	31.0
19	Cherbourg	NaN	7.2250	24.0

Cuối cùng, hãy vẽ biểu đồ ước tính mật độ hạt nhân cho cột tuổi ban đầu và cột tuổi chứa chế độ của các giá trị thay cho các giá trị bị thiếu.

Kịch bản 22:

```
plt.rcParams["figure.figsize"] = [8,6]

fig = plt.figure()
ax = fig.add_subplot(111)

titanic_data['tuổi'] .plot(loại='kde', ax=ax)

titanic_data['age_mode'] .plot(loại='kde', ax=ax, màu='đỏ')



dòng, nhãn = ax.get_legend_handles_labels()
ax.legend(dòng, nhãn, loc='tốt nhất')
```

Đầu ra:



Ưu điểm và nhược điểm

Việc quy kết danh mục thường xuyên dễ triển khai hơn trên các tập dữ liệu lớn. Phân phối danh mục thường xuyên không đưa ra bất kỳ giả định nào về dữ liệu và có thể được sử dụng trong sản xuất
môi trường.

Nhược điểm của việc quy kết danh mục thường xuyên là nó có thể biểu diễn quá mức danh mục xảy ra thường xuyên nhất trong trường hợp có quá nhiều giá trị bị thiếu trong tập dữ liệu gốc. Trong trường hợp các giá trị rất nhỏ trong tập dữ liệu gốc, việc quy kết danh mục thường xuyên có thể dẫn đến một nhãn mới chứa các giá trị hiếm
giá trị.

3.4.2. Thiếu sự quy kết danh mục

Việc quy ước danh mục bị thiếu tương tự như việc quy ước giá trị tùy ý. Trong trường hợp giá trị danh mục, việc quy ước giá trị bị thiếu sẽ thêm một danh mục tùy ý, ví dụ, thay thế giá trị bị thiếu bằng giá trị bị thiếu. Hãy xem một ví dụ về việc quy ước giá trị bị thiếu. Hãy tải tập dữ liệu Titanic và xem có cột danh mục nào chứa giá trị bị thiếu không.

Kịch bản 23:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns

plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

titanic_data = sns.load_dataset('titanic')

titanic_data = titanic_data[["embark_town", "tuổi", "giá vé"]]
titanic_data.đầu()
titanic_data.isnull().mean()
```

Đầu ra:

```
thị trấn_đi_vào 0.002245
tuổi          0,198653
giá vé          0,000000
Kiểu dữ liệu: float64
```

Đầu ra cho thấy cột embark_town là một cột danh mục cũng chứa một số giá trị bị thiếu. Chúng tôi sẽ áp dụng phép tính giá trị bị thiếu cho cột này.

Kịch bản 24:

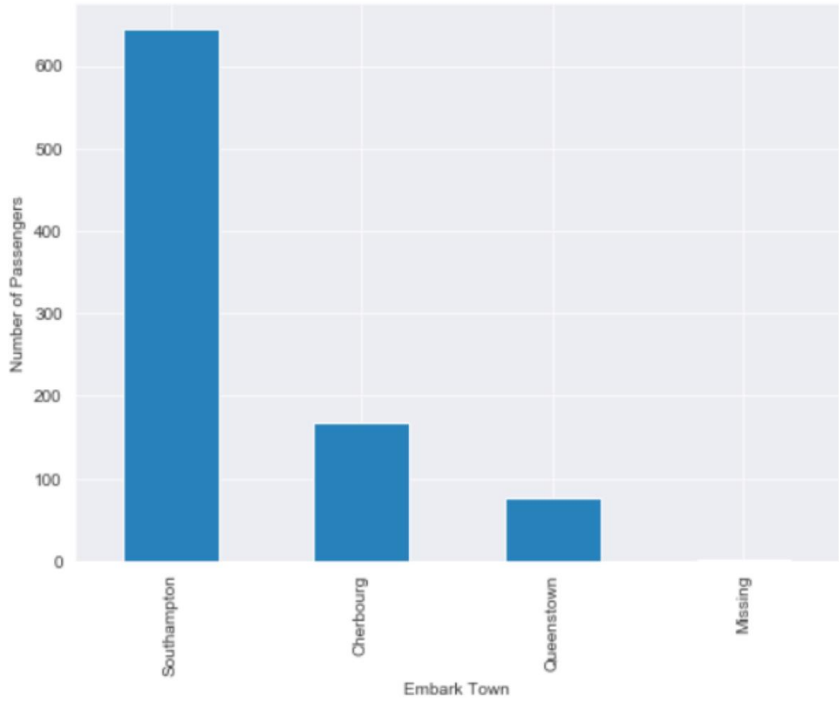
```
titanic_data.embark_town.fillna('Thiếu', inplace=Đúng)
```

Sau khi áp dụng phép tính giá trị bị thiếu, hãy vẽ biểu đồ thanh cho cột embark_town . Bạn có thể thấy rằng chúng ta có một biểu đồ rất nhỏ, gần như không đáng kể cho cột bị thiếu .

Kịch bản 25:

```
titanic_data.embark_town.value_counts().sort_
giá trị(tăng dần=Sai).plot.bar()
plt.xlabel('Vào thị trấn')
plt.ylabel('Số lượng hành khách')
```

Đầu ra:



Thời gian thực hành - Bài tập

Bây giờ đến lượt bạn. Hãy làm theo hướng dẫn trong bài tập bên dưới để kiểm tra hiểu biết của bạn về cách xử lý dữ liệu bị thiếu bằng Python. Câu trả lời cho những câu hỏi này được đưa ra ở cuối sách.

Bài tập 3.

Câu hỏi 1:

Nhược điểm chính của phương pháp quy ước giá trị trung bình và trung vị là gì?

A. Làm méo mó sự phân phối dữ liệu

B. Làm méo mó sự sai lệch của dữ liệu

C. Làm méo mó dữ liệu hiệp phương sai

D. Tất cả các câu trên

Câu hỏi 2:

Nên sử dụng phép quy kết nào khi dữ liệu không bị thiếu ngẫu nhiên?

A. Tính giá trị trung bình và trung vị

B. Quy kết giá trị tùy ý

C. Kết thúc phân phối

D. Quy kết giá trị bị thiếu

Câu hỏi 3:

Làm thế nào để tính toán kết thúc phân phối đuôi cho
phân phối chuẩn?

A. Quy tắc IQR

B. Trung bình x 3 Độ lệch chuẩn

C. Trung bình

D. Trung vị

Bài tập 3.

Thay thế các giá trị còn thiếu trong cột boong tàu Titanic
tập dữ liệu theo các danh mục xảy ra thường xuyên nhất trong đó
cột. Vẽ biểu đồ thanh cho cột sàn đã cập nhật .

4

Mã hóa dữ liệu danh mục

4.1 Giới thiệu

Các mô hình dựa trên thuật toán thống kê, chẳng hạn như học máy và học sâu, hoạt động với các con số. Tuy nhiên, một tập dữ liệu có thể chứa các biến số, biến danh mục, ngày giờ và biến hỗn hợp, như bạn đã thấy trong chương 2. Cần có một cơ chế để chuyển đổi dữ liệu danh mục thành dữ liệu số tương ứng để dữ liệu có thể được sử dụng để xây dựng các mô hình thống kê.

Các kỹ thuật được sử dụng để chuyển đổi dữ liệu số thành dữ liệu phân loại được gọi là các lược đồ mã hóa dữ liệu phân loại. Trong chương này, bạn sẽ thấy một số lược đồ mã hóa dữ liệu phân loại được sử dụng phổ biến nhất.

4.2 Một Encodi nóng

One hot encoding là một trong những lược đồ mã hóa theo danh mục được sử dụng phổ biến nhất. Trong one hot encoding, đối với mỗi giá trị duy nhất trong một cột theo danh mục, một cột mới sẽ được thêm vào. Số nguyên 1 được thêm vào cột tương ứng với nhãn gốc và tất cả các cột còn lại được điền bằng số không. Chúng ta hãy cùng xem một ví dụ rất đơn giản về one hot encoding.

Trong bảng sau, chúng ta có cột danh mục Quốc gia .
Cột này chứa ba giá trị duy nhất: Hoa Kỳ, Vương quốc Anh và Pháp.

Quốc gia	Mục tiêu
Hoa Kỳ	1
Anh quốc	0
Hoa Kỳ	1
Pháp	1
Hoa Kỳ	0
Anh quốc	0

Bảng sau đây chứa một phiên bản mã hóa nóng của bảng trên. Trong bảng sau đây, bạn có thể thấy ba cột đã được thêm vào, tức là USA, UK và FRANCE. Trong cột gốc, chúng ta có USA làm nhãn ở hàng đầu tiên của cột Country . Trong bảng mã hóa nóng mới thêm, chúng ta có 1 trong cột USA. Tương tự như vậy, bảng gốc

chứa Vương quốc Anh như nhãn ở hàng thứ hai. Trong một nóng

bảng được mã hóa, chúng ta có 1 ở hàng thứ hai cho cột Vương quốc Anh.

Hoa Kỳ	Anh quốc	Pháp	Mục tiêu
1	0	0	1
0	1	0	0
1	0	0	1
0	0	1	1
1	0	0	0
0	1	0	0

Trên thực tế, bạn chỉ cần N-1 cột trong một tập dữ liệu được mã hóa nóng cho một cột ban đầu chứa N nhãn duy nhất. Hãy xem bảng sau:

Anh quốc	Pháp	Mục tiêu
0	0	1
1	0	0
0	0	1
0	1	1
0	0	0
1	0	0

Trong bảng này, cột USA đã bị xóa. Tuy nhiên, chúng ta vẫn có thể nắm bắt thông tin mà cột đầu tiên chứa. Ví dụ, hàng mà cả hai cột UK và France đều chứa số không thực sự biểu thị rằng bản ghi này tương ứng với cột USA.

Chúng ta hãy cùng xem một mã hóa phổ biến thông qua ví dụ sau.
Thực hiện đoạn mã sau để tải xuống tập dữ liệu Titanic như chúng tôi đã làm ở các chương trước.

Kịch bản 1:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns

plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

titanic_data = sns.load_dataset('titanic')

titanic_data.đầu()
```

Đầu ra:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Hãy lọc khung dữ liệu titanic_data bằng cách xóa tất cả các cột ngoại trừ các cột sex, class và embark_town. Đây là các cột theo danh mục.

Kịch bản 2:

```
titanic_data = titanic_data[["sex", "class", "embark_town"]]
titanic_data.đầu()
```

Đầu ra:

	sex	class	embark_town
0	male	Third	Southampton
1	female	First	Cherbourg
2	female	Third	Southampton
3	female	First	Southampton
4	male	Third	Southampton

Hãy in các giá trị duy nhất trong ba cột trong khung dữ liệu titanic_data.

Kịch bản 3:

```
in(titanic_data['sex'].unique())
in(titanic_data['class'].unique())
in(titanic_data['embark_town'].unique())
```

Đầu ra:

```
['nam' 'nữ']
[Thứ ba, thứ nhất, thứ hai]
Thể loại (3, đối tượng): [Thứ ba, Thứ nhất, Thứ hai]
['Southampton' 'Cherbourg' 'Queenstown' ở đây]
```

Cách dễ nhất để chuyển đổi một cột thành một cột được mã hóa nóng là sử dụng phương thức get_dummies() của khung dữ liệu Pandas, như được hiển thị bên dưới:

Kịch bản 4:

```
nhập pandas dưới dạng pd
temp = pd.get_dummies(titanic_data['sex'])

temp. đầu()
```

Trong kết quả đầu ra, bạn sẽ thấy hai cột, một dành cho nam giới và một dành cho phụ nữ.

Đầu ra:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

Hãy hiển thị tên giới tính thực tế và tên được mã hóa nóng bỏng phiên bản cho cột giới tính trong cùng một khung dữ liệu.

Kịch bản 5:

```
pd.concat([titanic_data['sex'],
       pd.get_dummies(titanic_data['sex'])], axis=1).head()
```

Đầu ra:

	sex	female	male
0	male	0	1
1	female	1	0
2	female	1	0
3	female	1	0
4	male	0	1

Từ kết quả đầu ra ở trên, bạn có thể thấy rằng ở hàng đầu tiên, 1 có
đã được thêm vào cột nam vì giá trị thực tế trong
cột giới tính là nam. Tương tự, ở hàng thứ hai, thêm 1
vào cột nữ vì giá trị thực tế trong cột giới tính
là nữ.

Tương tự như vậy, chúng ta có thể chuyển đổi cột embark_town
thành một vector được mã hóa nóng như được hiển thị bên dưới:

Kịch bản 6:

```
nhập pandas dưới dạng pd
tạm thời = pd.get_dummies(titanic_data['embark_town'])

temp. đầu()
```

Đầu ra:

	Cherbourg	Queenstown	Southampton
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

Như bạn đã thấy trước đó, bạn có thể có N-1 cột được mã hóa nóng
các cột cho cột danh mục chứa N duy nhất
nhãn. Bạn có thể xóa cột đầu tiên được tạo bởi get_
phương thức dummies() bằng cách truyền True làm giá trị cho tham số
drop_first như hiển thị bên dưới:

Kịch bản 7:

```
nhập pandas dưới dạng pd
temp = pd.get_dummies(titanic_data['embark_town'], drop_first = Đúng)



temp. đầu()
```

Đầu ra:

	Queenstown	Southampton
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

Ngoài ra, bạn có thể tạo một cột được mã hóa nóng cho các giá trị null trong cột thực tế bằng cách truyền True làm giá trị cho tham số dummy_na.

Kịch bản 8:

```
nhập pandas dưới dạng pd
temp = pd.get_dummies(titanic_data['embark_town'], dummy_na = Đúng, drop_first
= Đúng)

temp. đầu()
```

Đầu ra:

	Queenstown	Southampton	NaN
0	0	1	0
1	0	0	0
2	0	1	0
3	0	1	0
4	0	1	0

Ưu điểm chính của one hot encoding là nó không đưa ra bất kỳ giả định nào về tập dữ liệu và tất cả các giá trị phân loại đều có thể được mã hóa thành công. Một nhược điểm lớn của phương pháp này là không gian đặc trưng có thể trở nên rất lớn vì một cột phân loại có thể có nhiều giá trị duy nhất.

4.3 Mã hóa nhãn

Trong mã hóa nhãn, nhãn được thay thế bằng số nguyên. Đây là lý do tại sao mã hóa nhãn cũng được gọi là mã hóa số nguyên. Hãy xem xét bảng sau:

Quốc gia	Mục tiêu
Hoa Kỳ	1
Anh quốc	0
Hoa Kỳ	1
Pháp	1
Hoa Kỳ	0
Anh quốc	0

Bảng trên đã được mã hóa nhãn như sau. Bạn có thể thấy rằng Hoa Kỳ đã được dán nhãn là 1, Vương quốc Anh đã được dán nhãn là 2, và Pháp được xếp hạng 3.

Quốc gia	Mục tiêu
1	1
2	0
1	1
3	1
1	0
2	0

Để thực hiện mã hóa nhãn, bạn có thể sử dụng LabelEncoder
lớp từ mô-đun sklearn.preprocessing, như được hiển thị bên dưới.
Bạn phải tạo một đối tượng của lớp label_encoder. Tiếp theo, bạn cần gọi phương thức fit() của đối tượng label_encoder và truyền cho nó cột phân loại của bạn. Cuối cùng, để chuyển đổi cột phân loại thành số, hãy gọi phương thức transform của đối tượng label_encoder và truyền cho nó cột phân loại.

Đoạn mã sau thực hiện mã hóa nhãn trên lớp
cột của tập dữ liệu Titanic.

Kịch bản 9:

```
# để mã hóa số nguyên bằng sklearn
từ sklearn.preprocessing nhập LabelEncoder

le = LabelEncoder()

le.fit(titanic_data['lớp'])

titanic_data['le_class'] = le.transform(titanic_data['class'])

titanic_data.đầu()
```

Đầu ra:

	sex	class	embark_town	le_class
0	male	Third	Southampton	2
1	female	First	Cherbourg	0
2	female	Third	Southampton	2
3	female	First	Southampton	0
4	male	Third	Southampton	2

Từ kết quả đầu ra ở trên, bạn có thể thấy lớp Third được gắn nhãn là 2, lớp First được gắn nhãn là 0, v.v. Điều quan trọng cần đề cập là mã hóa nhãn bắt đầu từ 0.

4.4 Mã hóa tần số

Trong mã hóa tần suất, mỗi nhãn duy nhất trong một cột danh mục được thay thế bằng tổng số hoặc tần suất của nó. Ví dụ, trong bảng sau, Hoa Kỳ xuất hiện ba lần, trong khi Vương quốc Anh và Pháp có số lần lần lượt là hai và một.

Quốc gia	Mục tiêu
Hoa Kỳ	1
Anh quốc	0
Hoa Kỳ	1
Pháp	1
Hoa Kỳ	0
Anh quốc	0

Sau khi mã hóa tần suất, cột Quốc gia trông như thế này.

Quốc gia	Mục tiêu
3	1
2	0
3	1
1	1
3	0
2	0

Hãy áp dụng mã hóa tần số trên cột embark_town
của tập dữ liệu Titanic. Cột chứa một số giá trị null
có thể được loại bỏ bằng cách sử dụng đoạn mã sau.

Kịch bản 10:

```
titanic_data.dropna(tại chỗ = True)
```

Tiếp theo, bạn cần gọi phương thức value_counts() trên cột phân loại, sau đó nối nó với to_dict()
cột để lấy số lượng cho mỗi nhãn duy nhất trong cột danh mục thực tế như được hiển thị bên dưới:

Kịch bản 11:

```
value_counts = titanic_data['embark_town'].value_counts().
to_dict()
in(giá_trị_đếm)
```

Đầu ra:

```
{'Southampton': 644, 'Cherbourg': 168, 'Queenstown': 77}
```

Cuối cùng, gọi phương thức map() và truyền cho nó từ điển chứa nhãn và số lượng.

Kịch bản 12:

```
titanic_data['embark_town'] = titanic_data['embark_town'].
bản đồ(giá_trị_đếm)
titanic_data.đầu()
```

Trong kết quả đầu ra, bạn có thể thấy rằng cột embark_town chứa tần suất của các nhãn thực tế.

Đầu ra:

	sex	class	embark_town	le_class
0	male	Third	644	2
1	female	First	168	0
2	female	Third	644	2
3	female	First	644	0
4	male	Third	644	2

Bạn cũng có thể thêm tần suất phần trăm bằng cách chia số lượng nhãn cho tổng số hàng như sau:

Kịch bản 13:

```
frequency_count = (titanic_data['embark_town'].value_counts() /
len(titanic_data) ).to_dict()
in(tần số_đếm)
```

Đầu ra:

```
{644: 0,7244094488188977, 168: 0,1889763779527559, 77:
0,08661417322834646}
```

Kịch bản 14:

```
titanic_data['embark_town'] = titanic_data['embark_town'].
bản đồ(tần số_đếm)
titanic_data.đầu()
```

Đầu ra:

	sex	class	embark_town	le_class
0	male	Third	0.724409	2
1	female	First	0.188976	0
2	female	Third	0.724409	2
3	female	First	0.724409	0
4	male	Third	0.724409	2

4.5 Encodi thứ tự

Trong mã hóa thứ tự, các nhãn được xếp hạng dựa trên mối quan hệ của chúng với mục tiêu. Ví dụ, trong Quốc gia
cột của bảng bên dưới, bạn có ba hàng trong đó Quốc gia là Hoa Kỳ cho ba hàng này và tổng số mục tiêu là 2. Do đó, giá trị trung bình mục tiêu sẽ là 2/3 = 0,66. Đối với Vương quốc Anh, giá trị này là 0 vì đối với cả hai lần xuất hiện của Vương quốc Anh, đều có số 0 trong cột Mục tiêu . Do đó, 0/2 = 0. Cuối cùng,
Pháp sẽ có giá trị là 1.

Quốc gia	Mục tiêu
Hoa Kỳ	1
Anh quốc	0
Hoa Kỳ	1
Pháp	1
Hoa Kỳ	0
Anh quốc	0

Tiếp theo, bạn xếp hạng các nhãn theo mức độ xuất hiện trung bình của chúng so với cột mục tiêu. Xếp hạng của chúng tôi sẽ là:

Pháp -> 1

Hoa Kỳ -> 0,66

Anh -> 0

Trong cột được mã hóa theo thứ tự, giá trị nhỏ nhất, tức là Vương quốc Anh sẽ được gán nhãn 0, Vương quốc Anh sẽ được gán nhãn 1, trong khi Pháp sẽ được gán nhãn 2 như hiển thị bên dưới:

Quốc gia	Mục tiêu
1	1
0	0
1	1
2	1
1	0
0	0

Hãy áp dụng mã hóa thứ tự trên cột lớp của tập dữ liệu Titanic.

Kịch bản 15:

```
titanic_data = sns.load_dataset('titanic')
titanic_data = titanic_data[["sex", "class", "embark_town", "survived"]]



titanic_data.groupby(['lớp'])['sống sót'].mean().sort_
giá trị()
```

Đầu ra:

```
lớp học
Thứ ba 0.242363
Thứ hai 0,472826
Đầu tiên 0.629630
Tên: survive, dtype: float64
```

Bạn có thể thấy rằng lớp First có giá trị trung bình cao nhất so với cột còn sống . Bạn có thể sử dụng bất kỳ cột nào khác làm cột mục tiêu . Tiếp theo, chúng ta tạo một từ điển trong đó các nhãn lớp được gán các nhãn số nguyên tương ứng. Cuối cùng, hàm map() được sử dụng để tạo một cột chứa các giá trị thứ tự, như được hiển thị bên dưới:

Kịch bản 16:

```
ordered_cats = titanic_data.groupby(['lớp'])['đã sống sót'].
mean().sort_values().index

cat_map= {k: i cho i, k trong enumerate(ordered_cats, 0)}

titanic_data['class_ordered'] = titanic_data['class'].map(cat_
bản đồ)

titanic_data.đầu()
```

Đầu ra:

	sex	class	embark_town	survived	class_ordered
0	male	Third	Southampton	0	0
1	female	First	Cherbourg	1	2
2	female	Third	Southampton	1	0
3	female	First	Southampton	1	2
4	male	Third	Southampton	0	0

Bạn có thể thấy rằng hầu hết hành khách đều ở hạng Nhất và được gắn nhãn cao nhất, tức là 2, v.v.

4.6 Encodi trung bình

Trong mã hóa trung bình, các nhãn được thay thế bằng các giá trị trung bình của chúng đối với các nhãn mục tiêu. Ví dụ, trong Quốc gia

cột của bảng bên dưới, bạn có ba hàng trong đó Quốc gia là Hoa Kỳ cho ba hàng này và tổng số mục tiêu là 2. Do đó, giá trị trung bình mục tiêu sẽ là 2/3 = 0,66. Đối với Vương quốc Anh, giá trị này là 0 vì đối với cả hai lần xuất hiện của Vương quốc Anh, đều có số 0 trong cột Mục tiêu . Do đó, 0/2 = 0. Cuối cùng,
Pháp sẽ có giá trị là 1.

Bảng thực tế:

Quốc gia	Mục tiêu
Hoa Kỳ	1
Anh quốc	0
Hoa Kỳ	1
Pháp	1
Hoa Kỳ	0
Anh quốc	0

Bảng mã hóa trung bình:

Quốc gia	Mục tiêu
0,66	1
0	0
0,66	1
1	1
0,66	0
0	0

Đoạn mã sau đây áp dụng mã hóa trung bình trên lớp cột của tập dữ liệu Titanic.

Kịch bản 17:

```
titanic_data.groupby(['lớp'])['sống sót'].mean()
```

Đầu ra:

```
lớp học
Đầu tiên 0.629630
Thứ hai 0,472826
Thứ ba 0.242363
Tên: survive, dtype: float64
```

Kịch bản 18:

```
mean_labels = titanic_data.groupby(['lớp'])['sống sót'].
trung bình().to_dict()
titanic_data['class_mean'] = titanic_data['class'].map(mean_
nhãn)
titanic_data.đầu()
```

Đầu ra:

	sex	class	embark_town	survived	class_ordered	class_mean
0	male	Third	Southampton	0	0	0.242363
1	female	First	Cherbourg	1	2	0.629630
2	female	Third	Southampton	1	0	0.242363
3	female	First	Southampton	1	2	0.629630
4	male	Third	Southampton	0	0	0.242363

Thời gian thực hành - Bài tập

Bây giờ đến lượt bạn. Hãy làm theo hướng dẫn trong bài tập bên dưới để kiểm tra hiểu biết của bạn về mã hóa dữ liệu theo danh mục bằng Python. Câu trả lời cho những câu hỏi này được đưa ra ở cuối sách.

Bài tập 4.

Câu hỏi 1:

Sơ đồ mã hóa nào thường dẫn đến số lượng cao nhất?
của các cột trong tập dữ liệu được mã hóa?

A. Mã hóa trung bình

B. Mã hóa thứ tự

C. Một mã hóa nóng

D. Tất cả các câu trên

Câu hỏi 2:

Thuộc tính nào được đặt thành True để xóa cột đầu tiên
từ các cột được mã hóa một nóng được tạo ra thông qua get_
phương thức dummies()?

A. thả_đầu_tiên

B. xóa_đầu_tiên

C. xóa_đầu_tiên

D. Không có câu nào ở trên

Câu hỏi 3:

Tổng số nhãn số nguyên trong mã hóa tần số là bao nhiêu?

A. Ít hơn một so với tổng số nhãn duy nhất trong
cột gốc

B. Bằng tổng số nhãn duy nhất trong bản gốc
cột

C.3

D. Không có câu nào ở trên

Bài tập 4.

Áp dụng mã hóa tần số cho cột lớp của Titanic
Bộ dữ liệu:

Phân tách dữ liệu

5.1 Giới thiệu

Trong chương trước, bạn đã nghiên cứu cách thực hiện mã hóa số của các giá trị phân loại. Trong chương này, bạn sẽ thấy cách chuyển đổi các giá trị số liên tục thành các giá trị rời rạc

khoảng thời gian.

Quá trình chuyển đổi các giá trị số liên tục như giá cả, độ tuổi và cân nặng thành các khoảng rời rạc được gọi là phân loại hoặc rời rạc.

Một trong những lợi thế chính của việc rời rạc hóa là nó có thể giúp xử lý các giá trị ngoại lai. Với việc rời rạc hóa, các giá trị ngoại lai có thể được đặt vào các khoảng đuôi cùng với các giá trị nội tại còn lại xảy ra ở đuôi. Rời rạc hóa đặc biệt hữu ích trong trường hợp bạn có sự phân phối dữ liệu bị lệch.

Có nhiều cách khác nhau để thực hiện phép rời rạc. Phép rời rạc có thể không giám sát và có giám sát. Trong không giám sát rời rạc, dữ liệu được rời rạc mà không phụ thuộc vào bất kỳ cột cụ thể nào trong tập dữ liệu. Mặt khác, trong phân rã có giám sát, dữ liệu được phân rã tùy thuộc vào giá trị trong một cột cụ thể trong tập dữ liệu.

Trong chương này, bạn sẽ thấy một số phương pháp phân biệt được sử dụng phổ biến nhất.

5.2 Phân rã độ rộng bằng nhau

Loại phương pháp phân biệt phổ biến nhất được sử dụng là phân biệt độ rộng cố định. Trong phân biệt độ rộng cố định,
chiều rộng hoặc kích thước của tất cả các khoảng vẫn giữ nguyên.
khoảng cách cũng được gọi là một thùng. Phân rã độ rộng bằng nhau là một loại kỹ thuật phân rã không giám sát.

Ví dụ, hãy xem xét một kịch bản trong đó bạn có một cột chứa giá của sản phẩm. Giá tối thiểu là 20, trong khi giá tối đa của một sản phẩm là 800. Nếu bạn muốn chuyển đổi các giá trị trong các cột giá thành 10 thùng hoặc 10 khoảng bằng cách sử dụng phép rời rạc có độ rộng bằng nhau, bạn chỉ cần trừ giá trị giá thấp hơn khỏi giá trị giá tối đa và sau đó chia kết quả cho tổng số khoảng.

Trong ví dụ của chúng tôi, với 10 khoảng thời gian, độ dài khoảng thời gian sẽ là 800 20/10 = 78. Trong kết quả đầu ra, bạn sẽ có 10 khoảng thời gian trong đó chiều rộng của mỗi khoảng sẽ là 78. Khoảng đầu tiên sẽ bắt đầu
từ 20 và kết thúc tại 20+78 = 98, v.v.

Bây giờ chúng ta hãy xem một ví dụ thực tế về phép rời rạc có độ rộng bằng nhau. Chúng tôi sẽ thực hiện phân biệt độ rộng bằng nhau trên cột giá
của bộ dữ liệu Diamonds. Bộ dữ liệu Diamonds xuất hiện
được tải sẵn thư viện Seaborn của Python và do đó, bạn
không cần phải tải xuống riêng biệt. Bạn chỉ cần có
thư viện Seaborn đã được cài đặt. Thực hiện tập lệnh sau để
tải tập dữ liệu Diamonds và hiển thị năm hàng đầu tiên của tập dữ liệu này.

Kịch bản 1:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns
nhập pandas dưới dạng pd
nhập numpy dưới dạng np
plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

diamond_data = sns.load_dataset('kim cương')

dữ liệu kim cương.đầu()
```

Đầu ra:

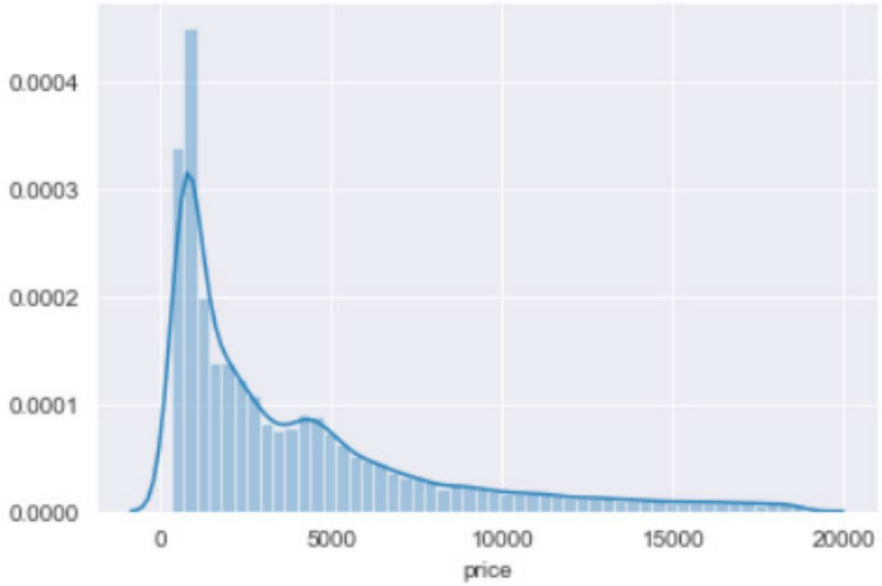
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Đầu ra cho thấy tập dữ liệu chứa 10 cột. Chúng ta sẽ chỉ thực hiện phân rã trên cột giá . Trước tiên, hãy vẽ biểu đồ histogram cho cột giá .

Kịch bản 2:

```
sns.distplot(dữ liệu kim cương['giá'])
```

Đầu ra:



Biểu đồ histogram cho cột giá cho thấy tập dữ liệu của chúng tôi bị lệch dương. Chúng ta có thể sử dụng phép rời rạc trên loại này phân phối dữ liệu.

Bây giờ chúng ta hãy tìm tổng phạm vi giá bằng cách trừ giá tối thiểu cho giá tối đa.

Kịch bản 3:

```
phạm vi giá = diamond_data['giá'].max() - diamond_
dữ liệu['giá'].min()
in(phạm vi giá)
```

Đầu ra:

```
18497
```

Mức giá là 18497. Chúng tôi sẽ tạo 10 khoảng có chiều rộng bằng nhau. Để tìm độ dài hoặc độ rộng của mỗi khoảng, chúng ta chỉ cần chia giá cho số khoảng.

Kịch bản 4:

```
phạm vi giá / 10
```

Đầu ra:

```
1849,7
```

Đầu ra hiển thị độ dài khoảng thời gian cho mỗi một trong 10 khoảng thời gian.

Giá tối thiểu sẽ được làm tròn đến giá sàn, trong khi giá tối đa sẽ được làm tròn đến giá trần. Giá sẽ được làm tròn đến giá trị số nguyên gần nhất. Đoạn mã sau
làm điều đó:

Kịch bản 5:

```
lower_interval = int(np.floor( diamond_data['price'].min()))
upper_interval = int(np.ceil(dữ liệu_kim_cương['giá'].max()))


interval_length = int(np.round(phạm vi giá / 10))

in(khoảng_khoảng_dưới)
in(khoảng_trên)
in(khoảng_dài_khoảng)
```

Đầu ra:

```
326
18823
1850
```

Tiếp theo, chúng ta hãy tạo 10 thùng cho tập dữ liệu của chúng ta. Để tạo thùng, chúng ta sẽ bắt đầu với giá trị tối thiểu và thêm khoảng thời gian bin hoặc
chiều dài của nó. Để có khoảng thứ hai, chiều dài của khoảng sẽ được thêm vào ranh giới trên của khoảng đầu tiên, v.v.
Đoạn mã sau đây tạo ra 10 thùng có chiều rộng bằng nhau.

Kịch bản 6:

```
total_bins = [i cho i trong phạm vi (khoảng_giữa_dưới, khoảng_giữa_trên
khoảng thời gian + độ dài khoảng thời gian, độ dài khoảng thời gian)]
in(tổng_số_thùng)
```

Đầu ra sau đây hiển thị ranh giới cho mỗi thùng.

Đầu ra:

```
[326, 2176, 4026, 5876, 7726, 9576, 11426, 13276, 15126, 16976, 18826]
```

Tiếp theo, chúng ta sẽ tạo nhãn chuỗi cho mỗi thùng. Bạn có thể cung cấp bất kỳ ghi tên vào nhãn thùng.

Kịch bản 7:

```
bin_labels = ['Bin_no_' + str(i) cho i trong phạm vi(1, len(tổng_
thùng))]
in(bin_labels)
```

Đầu ra:

```
['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_no_5',
'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9', 'Bin_no_10']
```

Đầu ra hiển thị nhãn bin cho tập dữ liệu của chúng tôi.

Bạn có thể tạo phương thức “cut()” của thư viện Pandas để chuyển đổi các giá trị cột liên tục thành các giá trị bin số. Bạn cần để truyền cột dữ liệu mà bạn muốn phân rã, cùng với khoảng cách giữa các thùng và nhãn thùng, như được hiển thị bên dưới.

Kịch bản 8:

```
diamond_data['price_bins'] = pd.cut(x=diamond_data['price'],
bins=tổng_số_thùng, nhãn=nhãn_thùng, include_lowest=Đúng)
kim cương_dữ_liệu.đầu(10)
```

Đầu ra:

	carat	cut	color	clarity	depth	table	price	x	y	z	price_bins
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	Bin_no_1
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	Bin_no_1
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	Bin_no_1
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	Bin_no_1
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	Bin_no_1
5	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48	Bin_no_1
6	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47	Bin_no_1
7	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53	Bin_no_1
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49	Bin_no_1
9	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39	Bin_no_1

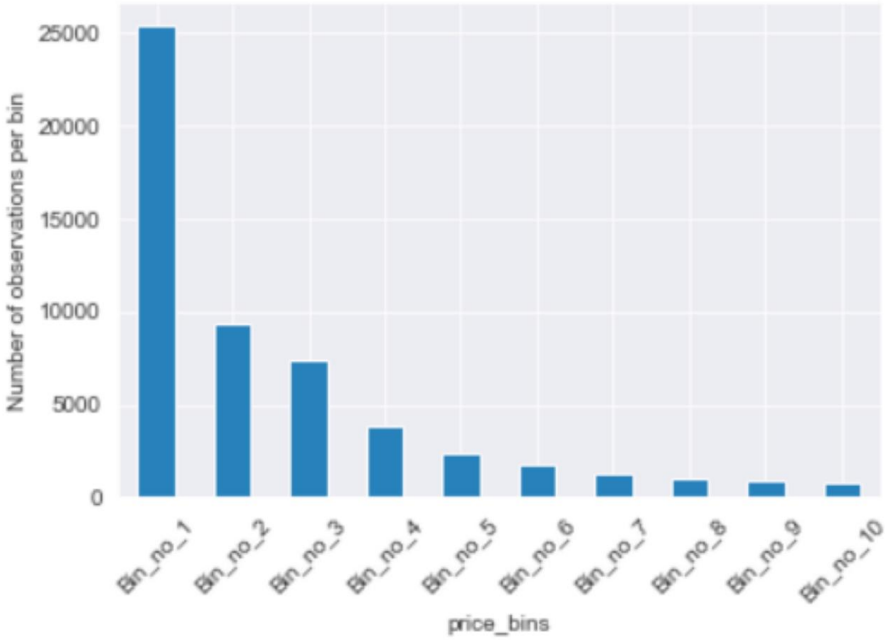
Trong kết quả đầu ra ở trên, bạn có thể thấy cột price_bins đã được thêm vào để hiển thị giá trị bin cho giá.

Tiếp theo, chúng ta hãy vẽ biểu đồ thanh thể hiện tần suất giá trong mỗi thùng.

Kịch bản 9:

```
diamond_data.groupby('price_bins')['price'].count().plot.bar()
plt.xticks(vòng quay=45)
```

Đầu ra:



Kết quả cho thấy giá của hầu hết các viên kim cương nằm ở
trong thùng đầu tiên hoặc khoảng thời gian đầu tiên.

5.3. Phân rã tần số bằng nhau

Trong rời rạc tần số bằng nhau, độ rộng của bin được điều chỉnh tự động theo cách mà mỗi bin chứa chính xác cùng một số bản ghi hoặc có cùng tần số. Do đó, tên là rời rạc tần số bằng nhau. Trong rời rạc tần số bằng nhau, khoảng bin có thể không giống nhau. Rời rạc tần số bằng nhau, giống như rời rạc độ rộng bằng nhau, là một kỹ thuật rời rạc có giám sát.

Hãy áp dụng phép phân biệt tần suất bằng nhau vào cột giá của tập dữ liệu Diamonds như chúng ta đã làm trước đó và xem chúng ta sẽ nhận được những thùng nào.

Đoạn mã sau đây tải xuống tập dữ liệu Diamonds.

Kịch bản 10:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns
nhập pandas dưới dạng pd
nhập numpy dưới dạng np
plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

diamond_data = sns.load_dataset('kim cương')

dữ liệu kim cương.đầu()
```

Đầu ra:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Để chuyển đổi một cột liên tục thành các bin rời rạc có tần suất bằng nhau, bạn có thể sử dụng hàm "qcut()". Hàm trả về các tứ phân vị, bằng với số khoảng thời gian được chỉ định cùng với các bin. Bạn phải truyền cột tập dữ liệu, số khoảng thời gian, nhãn làm tham số bắt buộc cho hàm "qcut()". Đoạn mã sau trả về 10 bin có tần suất bằng nhau cho cột giá của tập dữ liệu Diamonds. Tiếp theo, chúng ta tạo một khung dữ liệu hiển thị

giá thực tế và thông tin tứ phân vị.

Kịch bản 11:

```
giá_rời_rời, thùng = pd.qcut(dữ_liệu_kim_cương['giá'], 10, nhãn=Không có,
thùng_lại=Đúng, độ chính xác=3, trùng lặp='tăng')

pd.concat([giá_rời_rời, dữ_liệu_kim_cương['giá']], trục=1). đầu(10)
```

Đầu ra:

	price	price
0	(325.999, 646.0]	326
1	(325.999, 646.0]	326
2	(325.999, 646.0]	327
3	(325.999, 646.0]	334
4	(325.999, 646.0]	335
5	(325.999, 646.0]	336
6	(325.999, 646.0]	336
7	(325.999, 646.0]	337
8	(325.999, 646.0]	337
9	(325.999, 646.0]	338

Để xem khoảng cách giữa các thùng, chỉ cần in các thùng được trả về bởi hàm "qcut()" như hiển thị bên dưới:

Kịch bản 12:

```
in(thùng)
in(loại(thùng))
```

Đầu ra:

```
[ 326. 646. 837. 1087. 1698. 2401. 3465. 4662. 6301.2
 9821. 18823. ]
<class 'numpy.ndarray'>
```

Tiếp theo, hãy tìm số bản ghi trên mỗi thùng. Thực hiện đoạn mã sau:

Kịch bản 13:

```
giá_trị_đếm_rời_rời()
```

Đầu ra:

```
(325.999, 646.0] 5411
(1698.0, 2401.0] 5405
(837.0, 1087.0] 5396
(6301.2, 9821.0] 5395
(3465.0, 4662.0] 5394
(9821.0, 18823.0] 5393
(4662.0, 6301.2] 5389
(1087.0, 1698.0] 5388
(646.0, 837.0]     5385
(2401.0, 3465.0] 5384
Tên: giá, kiểu dữ liệu: int64
```

Từ đầu ra, bạn có thể thấy rằng tất cả các thùng có số lượng bản ghi ít nhiều giống nhau. Đây chính là những gì mà sự rời rạc tần số bằng nhau thực hiện, tức là tạo ra các thùng có số lượng bằng nhau
hồ sơ.

Tiếp theo, để tạo một khung dữ liệu Pandas chứa các thùng, chúng ta
Đầu tiên hãy tạo 10 nhãn vì chúng ta đã tạo 10 thùng.

Kịch bản 14:

```
bin_labels = ['Bin_no_' +str(i) cho i trong phạm vi(1,11)]
in(bin_labels)
```

Đầu ra:

```
['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_no_5',
'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9', 'Bin_no_10']
```

Để thực hiện phân loại, chúng ta có thể sử dụng lại thư viện Pandas
Phương thức “cut()” như được hiển thị bên dưới:

Kịch bản 15:

```
diamond_data['price_bins'] = pd.cut(x=diamond_data['price'], bins=bins,
labels=bin_labels, include_lowest=True)
kim cương_dữ_liệu.đầu(10)
```

Đầu ra:

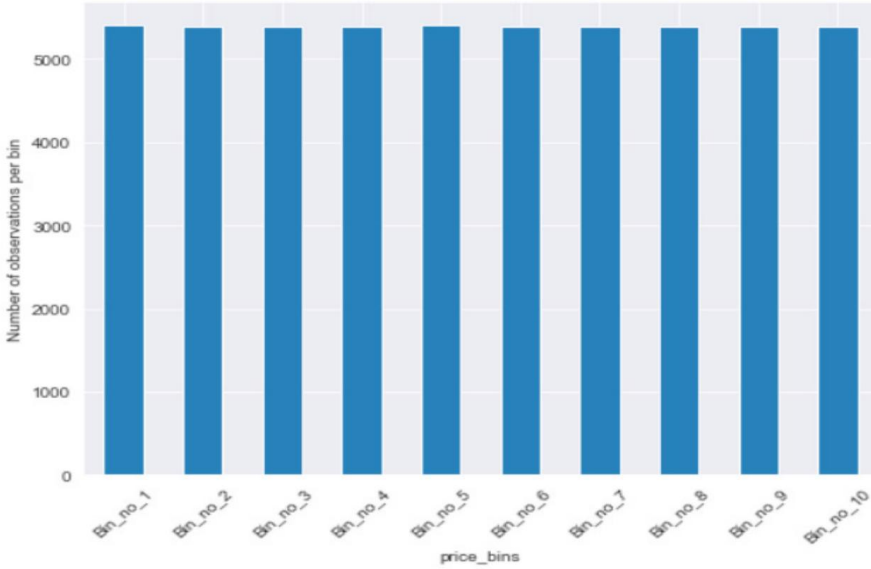
	carat	cut	color	clarity	depth	table	price	x	y	z	price_bins
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	Bin_no_1
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	Bin_no_1
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	Bin_no_1
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	Bin_no_1
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	Bin_no_1
5	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48	Bin_no_1
6	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47	Bin_no_1
7	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53	Bin_no_1
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49	Bin_no_1
9	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39	Bin_no_1

Trong kết quả đầu ra ở trên, bạn có thể thấy một cột mới, tức là price_
thùng. Cột này chứa các nhãn thùng rời rạc có tần số bằng nhau.
Cuối cùng, chúng ta có thể vẽ biểu đồ thanh hiển thị tần suất bản ghi trên
mỗi thùng.

Kịch bản 16:

```
diamond_data.groupby('price_bins')['price'].count().plot.bar()
plt.xticks(vòng quay=45)
```

Đầu ra:



Bạn có thể thấy rằng số lượng bản ghi gần như giống nhau đối với

tất cả các thùng rác.

5.4. Phân rã K-Means

Phân rã K-means là một kỹ thuật phân rã không giám sát khác dựa trên thuật toán K-means.

Mô tả tóm tắt về thuật toán K-Means được đưa ra dưới đây:

1. Ban đầu, K cụm điểm dữ liệu ngẫu nhiên được

   được tạo ra, trong đó K là số lượng thùng hoặc khoảng.

2. Mỗi điểm dữ liệu được liên kết với trung tâm cụm gần nhất.

3. Trung tâm của tất cả các cụm được cập nhật dựa trên các điểm dữ liệu liên quan.

Hãy sử dụng phép rời rạc K-Means để rời rạc cột giá của tập dữ liệu Diamonds. Tập lệnh

sau đây nhập

tập dữ liệu.

Kịch bản 17:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns
nhập pandas dưới dạng pd
nhập numpy dưới dạng np
từ sklearn.preprocessing nhập KBinsDiscretizer

plt.rcParams["figure.figsize"] = [8,6]
sns.set_style("lưới tối")

diamond_data = sns.load_dataset('kim cương')

dữ liệu kim cương.đầu()
```

Đầu ra:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Tiếp theo, để thực hiện phân biệt, bạn cần gọi phương thức “KBinsDiscretizer()” và truyền cho nó số lượng thùng. Trên đối tượng được trả về bởi phương thức “KBinsDiscretizer()” , bạn cần gọi phương thức “fit()” và truyền cho nó cột giá, như được hiển thị bên dưới:

Kịch bản 18:

```
rời rạc = KBinsDiscretizer(n_bins=10, mã hóa='thứ tự', chiến lược='kmeans')



rời rạc.fit(diamond_data[['price']])
```

Tiếp theo, bạn có thể truy cập các thùng được tạo thông qua cụm K-means bằng cách sử dụng thuộc tính “bin_edges”.

Kịch bản 19:

```
khoảng cách = discretization.bin_edges_.tolist()
in(khoảng cách)
```

Đầu ra:

```
[mảng([ 326.       , 1417.67543928, 2627.50524806, 3950.3762392,
    5441.70606939, 7160.05893161, 9140.61465361,
11308.37609661,
    13634.55462656, 16130.22549621, 18823.        ])]
```

Hãy tạo một danh sách các thùng được tạo thông qua phương pháp phân rã K-means.

Kịch bản 20:

```
khoảng cách = [ 326.           , 1417.67543928, 2627.50524806,
3950.3762392       ,
    5441.70606939, 7160.05893161, 9140.61465361,
11308.37609661,
    13634.55462656, 16130.22549621, 18823.        ]
```

Đoạn mã sau đây tạo danh sách nhãn cho mỗi thùng.

Kịch bản 21:

```
bin_labels = ['Bin_no_' +str(i) cho i trong phạm vi(1,11)]
in(bin_labels)
```

Đầu ra:

```
['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_
no_5', 'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9',
'Bin_no_10']
```

Cuối cùng, bạn có thể sử dụng phương thức cắt của khung dữ liệu Pandas để tạo một cột mới chứa các thùng cho cột giá .

Kịch bản 22:

```
diamond_data['price_bins'] = pd.cut(x=diamond_data['price'], bins=khoảng
cách, nhãn=bin_labels, include_lowest=True)
kim cương_dữ_liệu.đầu(10)
```

Đầu ra:

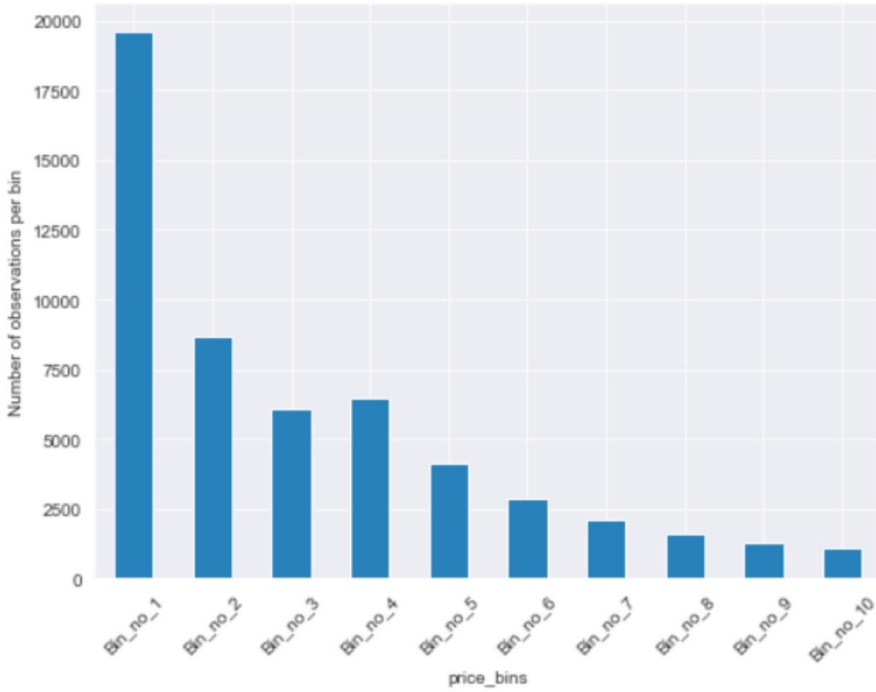
	carat	cut	color	clarity	depth	table	price	x	y	z	price_bins
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	Bin_no_1
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	Bin_no_1
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	Bin_no_1
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	Bin_no_1
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	Bin_no_1
5	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48	Bin_no_1
6	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47	Bin_no_1
7	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53	Bin_no_1
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49	Bin_no_1
9	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39	Bin_no_1

Đoạn mã sau đây vẽ biểu đồ thanh hiển thị tần suất bản ghi trên mỗi thùng.

Kịch bản 23:

```
diamond_data.groupby('price_bins')['price'].count().plot.bar()
plt.xticks(vòng quay=45)
```

Đầu ra:



5.5. Cây quyết định rời rạc

Phân biệt cây quyết định là một loại thuật toán phân biệt có giám sát. Trong phân biệt cây quyết định, các thùng
được tạo ra dựa trên các giá trị trong một số cột khác. Hãy
Trước tiên hãy tải xuống bộ dữ liệu Diamonds.

Kịch bản 24:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns
nhập pandas dưới dạng pd
nhập numpy dưới dạng np
từ sklearn.preprocessing nhập KBinsDiscretizer
từ sklearn.tree nhập DecisionTreeClassifier

sns.set_style("lưới tối")

diamond_data = sns.load_dataset('kim cương')

dữ liệu kim cương.đầu()
```

Đầu ra:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Trong phân loại cây quyết định, chúng tôi không chỉ định số lượng thùng hoặc khoảng cách. Thay vào đó, cây quyết định xác định
số lượng thùng tối ưu.

Để thực hiện phân biệt cây quyết định, bạn có thể sử dụng
Lớp "DecisionTreeClassifier" từ "sklearn.tree"
module. Bạn cần gọi phương thức "fit()" trên lớp và truyền tên cột liên tục và cột mà bạn muốn dựa vào đó để tạo các thùng của mình.

Ví dụ, đoạn mã sau đây tạo các thùng cho cột giá của tập dữ liệu Kim cương, dựa trên các giá trị trong cột cắt .

Kịch bản 25:

```
tree_model = Phân loại cây quyết định (độ sâu tối đa = 3)

tree_model.fit(dữ liệu_kim_cương['giá'].to_frame(), dữ
liệu_kim_cương['cắt'])

diamond_data['price_tree']= tree_model.predict_proba(diamond_data['price'].to_frame())
[:,1]

dữ liệu kim cương.đầu()
```

Đầu ra:

	carat	cut	color	clarity	depth	table	price	x	y	z	price_tree
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	0.127435
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	0.127435
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	0.127435
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	0.127435
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	0.127435

Bạn có thể tìm các giá trị xác suất duy nhất trong cột price_tree bằng cách sử dụng tập lệnh sau:

Kịch bản 26:

```
diamond_data['price_tree'].unique()
```

Đầu ra:

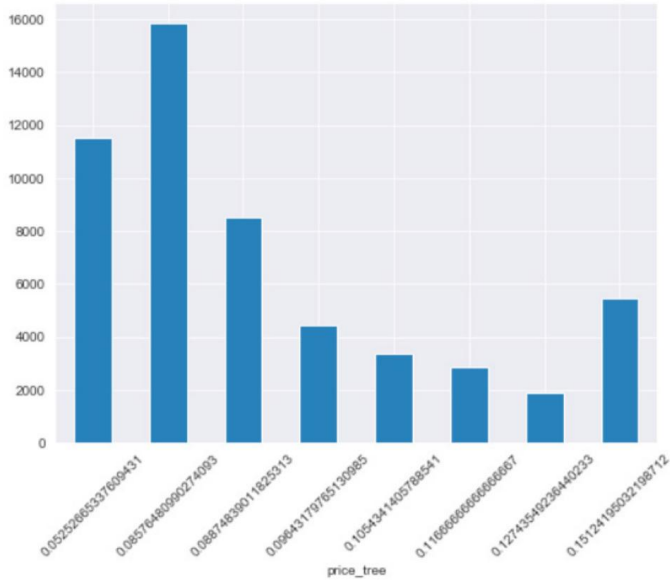
```
mảng([0.12743549, 0.10543414, 0.0964318, 0.11666667, 0.15124195,

     0,08576481, 0,05252665, 0,08874839])
```

Đoạn mã sau đây biểu thị tần suất các bản ghi trên mỗi thùng.

Kịch bản 27:

```
diamond_data.groupby(['price_tree'])['price'].count().plot.bar()

plt.xticks(vòng quay=45)
```

Đầu ra:



5.6. Phân biệt tùy chỉnh

Cuối cùng, bạn có thể thực hiện phân biệt tùy chỉnh của một cột liên tục bằng cách truyền các giá trị bin tùy chỉnh. Hãy rời rạc cột tip của tập dữ liệu Tips. Sau đây tập lệnh tải xuống tập dữ liệu Mẹo.

Kịch bản 28:

```
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns
nhập pandas dưới dạng pd
nhập numpy dưới dạng np
từ sklearn.preprocessing nhập KBinsDiscretizer
từ sklearn.tree nhập DecisionTreeClassifier

sns.set_style("lưới tối")

tips_data = sns.load_dataset('tips')

mẹo_dữ_liệu.đầu()
```