Đầu ra:

```
10 lớn hơn 5
```

Câu lệnh IF-Elif

Câu lệnh if-elif hữu ích khi bạn phải đánh giá nhiều điều kiện. Ví dụ, trong ví dụ sau, đầu tiên chúng ta kiểm tra nếu 5 > 10, kết quả trả về là false. Tiếp theo, câu lệnh elif đánh giá điều kiện 8 < 4, kết quả cũ ng

trả về false. Do đó, khối mã theo sau else cuối cùng
câu lệnh được thực thi.

Kịch bản 10:

```
#if-elif và else

nếu 5 > 10:
  print("5 lớn hơn 10")
nếu 8 < 4:
  print("8 nhỏ hơn 4")
khác:
  print("5 không lớn hơn 10 và 8 không nhỏ hơn 4")
```

Đầu ra:

```
5 không lớn hơn 10 và 8 không nhỏ hơn 4
```

1.3.5. Các câu lệnh lặp lại

Các câu lệnh lặp, còn được gọi là vòng lặp, được sử dụng để thực thi lặp đi lặp lại một đoạn mã nhất định. Có hai loại câu lệnh lặp chính trong Python.

a. Vòng lặp For

b. Vòng lặp While

Vòng lặp For

Vòng lặp for được sử dụng để thực thi lặp đi lặp lại một đoạn mã trong một số lần nhất định. Bạn nên sử dụng vòng lặp for khi bạn biết chính xác số lần lặp lại hoặc số lần lặp lại mà bạn muốn chạy mã của mình. Vòng lặp for lặp lại một tập hợp các mục. Trong ví dụ sau, chúng ta tạo một tập hợp gồm năm số nguyên bằng phương thức range(). Tiếp theo, vòng lặp for lặp lại năm lần và in ra từng số nguyên trong

bộ sưu tập.

Kịch bản 11:

```
mục = phạm vi(5)
cho mục trong mục:
in(mục)
```

Đầu ra:

```
0
1
2
3
4
```

Vòng lặp While

Vòng lặp while tiếp tục thực thi một đoạn mã nhất định trừ khi điều kiện đánh giá trở thành sai. Ví dụ, while

Vòng lặp trong đoạn mã sau sẽ tiếp tục thực thi trừ khi biến c lớn hơn 10.

Kịch bản 12:

```
c = 0
trong khi c < 10:
  in(c)
c = c +1
```

Đầu ra:

```
0
1
2
3
4
5
6
7
8
9
```

1.3.6. Chức năng

Các hàm, trong bất kỳ ngôn ngữ lập trình nào, được sử dụng để thực hiện đoạn mã cần được thực thi
nhiều lần ở các vị trí khác nhau trong mã. Trong đó
trong một số trường hợp, thay vì viết đi viết lại những đoạn mã dài, bạn chỉ cần định nghĩa một hàm chứa đoạn mã đó, sau đó bạn có thể gọi hàm đó ở bất cứ đâu bạn muốn

trong mã.

Từ khóa def được sử dụng để tạo một hàm trong Python, theo sau là tên hàm và dấu ngoặc đơn mở và đóng.

Sau khi một hàm được định nghĩa, bạn phải gọi hàm đó để thực thi mã bên trong thân hàm. Để gọi hàm, bạn chỉ cần chỉ định tên hàm, theo sau là dấu ngoặc đơn mở và đóng. Trong tập lệnh sau, chúng ta tạo một hàm có tên là myfunc, hàm này in một câu lệnh đơn giản trên bảng điều khiển bằng phương thức print().

Kịch bản 13:

```
định nghĩa myfunc():
   print("Đây là một hàm đơn giản")

### gọi hàm
hàm myfunc()
```

Đầu ra:

```
Đây là một chức năng đơn giản
```

Bạn cũ ng có thể truyền giá trị cho một hàm. Các giá trị được truyền bên trong dấu ngoặc đơn của lệnh gọi hàm. Tuy nhiên, bạn cũ ng phải chỉ định tên tham số trong định nghĩa hàm. Trong tập lệnh sau, chúng tôi định nghĩa một hàm có tên là myfuncparam(). Hàm chấp nhận một tham số, tức là num.

Giá trị được truyền vào trong dấu ngoặc đơn của lệnh gọi hàm sẽ được lưu trữ trong biến num này và sẽ được in ra bởi lệnh print() phương thức bên trong phương thức myfuncparam().

Kịch bản 14:

```
def myfuncparam(số):
   print("Đây là một hàm có giá trị tham số: "+num)

### gọi hàm
myfuncparam("Tham số 1")
```

Đầu ra:

```
Đây là một hàm có giá trị tham số:Tham số 1
```

Cuối cùng, một hàm cũ ng có thể trả về giá trị cho lệnh gọi hàm. Để để làm như vậy, bạn chỉ cần sử dụng từ khóa return, theo sau là giá trị mà bạn muốn trả về. Trong tập lệnh sau, Hàm myreturnfunc() trả về một giá trị chuỗi cho lệnh gọi chức năng.

Kịch bản 15:

```
định nghĩa myreturnfunc():
   trả về "Hàm này trả về một giá trị"


giá trị = myreturnfunc()
in(val)
```

Đầu ra:

```
Hàm này trả về một giá trị
```

1.3.7. Đối tượng và Lớp

Python hỗ trợ lập trình hướng đối tượng (OOP). Trong OOP, bất kỳ thực thể nào có thể thực hiện một số chức năng và có một số thuộc tính đều được triển khai dưới dạng đối tượng.

Ví dụ, một chiếc ô tô có thể được triển khai như một đối tượng vì ô tô có một số thuộc tính như giá cả, màu sắc, kiểu dáng và có thể thực hiện một số chức năng như lái xe, sang số, dừng xe, v.v.

Tương tự như vậy, một loại trái cây cũ ng có thể được triển khai như một đối tượng vì trái cây có giá, tên và bạn có thể ăn trái cây, trồng trái cây và thực hiện các chức năng với trái cây.

Để tạo một đối tượng, trước tiên bạn phải định nghĩa một lớp. Đối với
Ví dụ, trong ví dụ sau, một lớp Fruit đã được
đã định nghĩa. Lớp có hai thuộc tính name và price, và một phương thức eat_fruit(). Tiếp theo, chúng ta tạo một đối tượng f của lớp Fruit rồi gọi phương thức eat_fruit() từ đối tượng f. Chúng ta cũ ng truy cập các thuộc tính name và price của đối tượng f và in chúng ra bảng điều khiển.

Kịch bản 16:

```
lớp Trái cây:

   tên = "táo" giá =
   10

   def eat_fruit(self):
      print("Trái cây đã được ăn")


f = Fruit()
f.eat_fruit()
in(f.name)
in(f.price)
```

Đầu ra:

```
Trái cây đã được ăn
táo 10
```

Một lớp trong Python có thể có một phương thức đặc biệt gọi là constructor. Tên của phương thức constructor trong Python là __init__(). Constructor được gọi bất cứ khi nào một đối tượng của một lớp được tạo ra. Hãy xem ví dụ sau để thấy

người xây dựng đang hành động.

Kịch bản 17:

```
lớp Trái cây:

   tên = "táo" giá =
   10

   def __init__(self, tên_quả, giá_quả):
      Tên trái cây = tên_trái_cây
      Giá trái cây = giá_trái_cây

   def eat_fruit(self):
      print("Trái cây đã được ăn")
```

```
f = Trái cây("Cam", 15)
f.ăn_trái_quả()
in(f.name)
in(f.giá)
```

Đầu ra:

```
Trái cây đã được ăn
Quả cam
15
```

Đọc thêm - Python [1]

Để tìm hiểu thêm về Python, vui lòng kiểm tra Tài liệu chính thức của Python 3 . Hãy làm quen với việc tìm kiếm và đọc tài liệu này. Đây là nguồn kiến thức tuyệt vời.

1.4 Các thư viện khác nhau cho Tiền xử lý dữ liệu

Do tầm quan trọng ngày càng tăng của việc xử lý trước dữ liệu, một số thư viện Python đã được phát triển. Một số thư viện này đã được xem xét ngắn gọn trong phần này.

1.4.1.NumPy

NumPy là một trong những thư viện được sử dụng phổ biến nhất cho tính toán số và khoa học. NumPy cực kỳ nhanh và hỗ trợ nhiều lĩnh vực toán học như đại số tuyến tính, hình học, v.v. Học NumPy là vô cùng quan trọng nếu bạn có kế hoạch theo đuổi sự nghiệp khoa học dữ liệu và xử lý dữ liệu trước.

1.4.2. Học Scikit

Scikit learn, còn được gọi là sklearn, là một thư viện cực kỳ hữu ích cho việc học máy trong Python. Sklearn chứa nhiều

các mô-đun có thể được sử dụng để thực hiện các tác vụ xử lý trước dữ liệu như kỹ thuật tính năng, mở rộng tính năng, phát hiện giá trị ngoại lai, rời rạc, v.v. Bạn sẽ sử dụng Sklearn rất nhiều trong cuốn sách này.

Do đó, bạn nên học sklearn trước khi bắt đầu viết mã bằng cuốn sách này.

1.4.3. Chương trình Matplotlib

Trực quan hóa dữ liệu là bước tiền xử lý quan trọng trước khi xử lý dữ liệu. Trước khi thực sự áp dụng các kỹ thuật xử lý dữ liệu trên dữ liệu, bạn nên biết dữ liệu trông như thế nào, phân phối của một biến nhất định là gì, v.v. Matplotlib

là tiêu chuẩn thực tế cho việc trực quan hóa dữ liệu tĩnh trong Python.

1.4.4. Sinh ra ở biển

Thư viện Seaborn được xây dựng trên thư viện Matplotlib và chứa tất cả các khả năng vẽ đồ thị của Matplotlib. Tuy nhiên, với Seaborn, bạn có thể vẽ đồ thị đẹp mắt và thẩm mỹ hơn nhiều với sự trợ giúp của các kiểu mặc định và bảng màu của Seaborn.

1.4.5. Gấu trúc

Thư viện Pandas, giống như Seaborn, dựa trên thư viện Matplotlib và cung cấp các tiện ích có thể được sử dụng để vẽ các loại biểu đồ tĩnh khác nhau trong một dòng mã. Với Pandas, bạn có thể nhập dữ liệu ở nhiều định dạng khác nhau như CSV (Comma Separated View) và TSV (Tab Separated View) và có thể vẽ nhiều hình ảnh dữ liệu khác nhau thông qua các nguồn dữ liệu này.

Đọc thêm – Thư viện tiền xử lý dữ liệu

Để tìm hiểu thêm về các thư viện xử lý dữ liệu trước cho Python, hãy kiểm tra các liên kết sau:

Số lượng [1] (https://numpy.org/)
Học Scikit [2] (https://scikit-learn.org/stable/)
Matplotlib [3] (https://matplotlib.org/)
Sinh ra ở biển [4] (https://seaborn.pydata.org/)
Gấu trúc [5] (https://pandas.pydata.org/)

Thời gian thực hành – Bài tập

Bây giờ đến lượt bạn. Thực hiện theo hướng dẫn trong các bài tập bên dưới để kiểm tra sự hiểu biết của bạn về trực quan hóa dữ liệu nâng cao với Matplotlib. Câu trả lời cho những câu hỏi này được đưa ra ở cuối sách.

Bài tập 1.

Câu hỏi 1:

Nên sử dụng vòng lặp nào khi bạn muốn thực thi nhiều lần một mã lệnh với số lần cụ thể?

Một vòng lặp For

B Vòng lặp While

C Cả A và B

D Không có câu nào ở trên

Câu hỏi 2:

Số lượng giá trị tối đa mà một hàm có thể nhận được là bao nhiêu?

trả về trong Python?

Một Giá Trị Duy Nhất

B Giá trị kép

C Nhiều hơn hai giá trị

D Không có

Câu hỏi 3:

Trong các toán tử thành viên sau đây, toán tử nào được Python hỗ trợ?

Một Trong

B Ra ngoài

C Không Có Trong

D Cả A và C

Bài tập 1.

In bảng số nguyên 9 bằng cách sử dụng vòng lặp while:

§ Tài liệu tham khảo

1. https://numpy.org/
2. https://scikit-learn.org/
3. https://matplotlib.org/
4. https://seaborn.pydata.org/index.html
5. https://pandas.pydata.org/

Hiểu về các kiểu dữ liệu

2.1 Giới thiệu

Một tập dữ liệu có thể chứa các biến có nhiều loại khác nhau tùy thuộc vào dữ liệu mà chúng lưu trữ. Điều quan trọng là phải biết các loại dữ liệu khác nhau mà một biến có thể lưu trữ vì cần có các kỹ thuật khác nhau để xử lý dữ liệu có nhiều loại khác nhau. Trong chương này, bạn sẽ thấy các loại dữ liệu khác nhau mà bạn có thể
gặp phải.

2.1.1. Biến là gì?

Trước khi tìm hiểu về biến có nhiều kiểu dữ liệu khác nhau, trước tiên chúng ta hãy định nghĩa biến là gì.

“Biến là một thực thể lưu trữ giá trị tương ứng với một đặc điểm, số lượng hoặc một con số có thể đếm được
hoặc được đo lường.”

Sau đây là một số ví dụ về biến:

- Tên (Jon, Mike, Alen, v.v.)
- Tuổi (50, 60, 52, v.v.)
- Giới tính (Nam, Nữ, v.v.)
- Quốc tịch (Mỹ, Anh, Ấn Độ, v.v.)

•Ngày sinh (10-10-1990, 12-03-1983, v.v.)

Bạn có thể thấy rằng một biến có thể chứa dữ liệu có nhiều kiểu khác nhau, bao gồm số, chuỗi, ngày tháng, v.v.

2.1.2. Kiểu dữ liệu

Nhìn chung, dữ liệu có thể được phân loại thành các loại sau:

1. Dữ liệu số
2. Dữ liệu phân loại
3. Biến Ngày và Giờ
4. Biến hỗn hợp

Dữ liệu số có thể được chia thành hai loại: Dữ liệu rời rạc và Dữ liệu liên tục. Mặt khác, dữ liệu danh mục có thể được chia thành ba loại chính: Dữ liệu danh nghĩa, Dữ liệu thứ tự và Dữ liệu ngày/giờ.

2.2 Số Da

Dữ liệu số là loại dữ liệu bao gồm các con số.
Dữ liệu số có thể được chia thành hai loại: Rời rạc
và Liên tục.

Các biến dữ liệu rời rạc là những biến lưu trữ một
số nguyên. Ví dụ, số lượng trẻ em của một
người có thể là 2, 3, 4 hoặc bất kỳ số rời rạc nào. Bạn không thể nói rằng anh ta có 2 đứa con rưỡi. Tương tự như vậy, số xe mà một người sở hữu hoặc số anh chị em của một người là
cũ ng là một số rời rạc. Tất cả những số này đều là các giá trị số rời rạc.

Mặt khác, các biến dữ liệu liên tục là các biến giữ các giá trị liên tục, chẳng hạn như số tiền trong tài khoản ngân hàng của một người, có thể là 2560,04 đô la. Tương tự như vậy,

trọng lượng của một loại rau, có thể là 2,5kg, hoặc giá của một chiếc ô tô, có thể là 4057,34 đô la.

Bây giờ chúng ta hãy cùng khám phá một số ví dụ về dữ liệu số từ tập dữ liệu thực tế, tức là Tập dữ liệu Titanic.

Bộ dữ liệu Titanic là một bộ dữ liệu nổi tiếng có chứa thông tin về những hành khách có mặt trên con tàu Titanic không may bị chìm vào năm 1912.

Điều quan trọng cần đề cập là trước khi bạn chạy tập lệnh để nhập tập dữ liệu Titanic, hoặc bất kỳ tập dữ liệu nào, bạn cần cài đặt một số thư viện Python. Để làm như vậy, bạn phải để thực hiện lệnh sau trên dấu nhắc lệnh của bạn hoặc Dấu nhắc Anaconda, tùy thuộc vào môi trường Python mà bạn đã cài đặt trên hệ thống của mình.

```
pip cài đặt pandas
pip cài đặt numpy
pip cài đặt matplotlib
pip cài đặt seaborn
```

Tiếp theo, thực thi đoạn mã sau để nhập tập dữ liệu Titanic:

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()
```

Tập lệnh trên đầu tiên nhập Matplotlib và các thư viện Seaborn. Sau đó, nó tăng kích thước của biểu đồ mặc định và sau đó đặt kiểu lưới thành tối. Cuối cùng, tập dữ liệu trên sử dụng

phương thức load_dataset() của thư viện Seaborn để nhập tập dữ liệu Titanic tích hợp. Cuối cùng, năm hàng đầu tiên của tập dữ liệu Titanic đã được hiển thị. Đây là đầu ra của tập lệnh
bên trên:

	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

Bạn có thể thấy rằng tập dữ liệu chứa thông tin về
giới tính, độ tuổi, giá vé, hạng, v.v. của hành khách. Bây giờ chúng ta hãy xác định các cột chứa dữ liệu rời rạc và liên tục
giá trị số.

2.2.1. Dữ liệu rời rạc

Hãy xem cột pclass . Cột này chứa thông tin về hạng ghế mà hành khách đã đi.

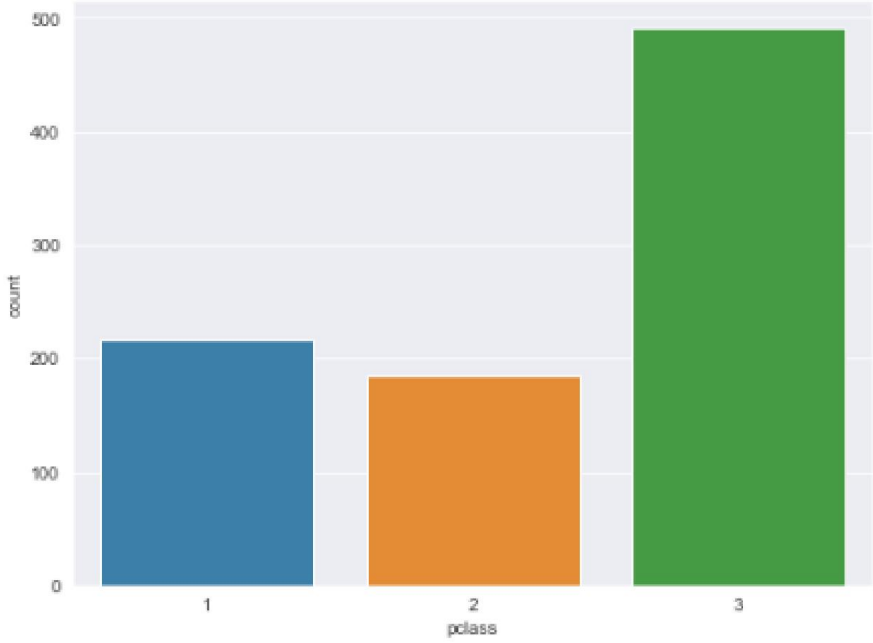
Cột pclass có thể có ba giá trị khả thi là 1, 2 và 3 tương ứng với hạng 1, 2 và 3 mà hành khách đã đi. Vì hành khách không thể đi hạng 1,5 hoặc hạng 2,5, nên chúng ta có thể nói rằng cột pclass chứa các giá trị số rời rạc. Hãy vẽ một biểu đồ thanh hiển thị số lượng hành khách đi trong mỗi hạng trong ba hạng:

Kịch bản 2:

```
sns.countplot(x='pclass', dữ liệu=titanic_data)
```

Kết quả cho thấy có khoảng 200 hành khách đã đi du lịch
hạng nhất trong khi phần lớn hành khách đi hạng 3 trên tàu Titanic.

Đầu ra:



2.2.2. Dữ liệu liên tục

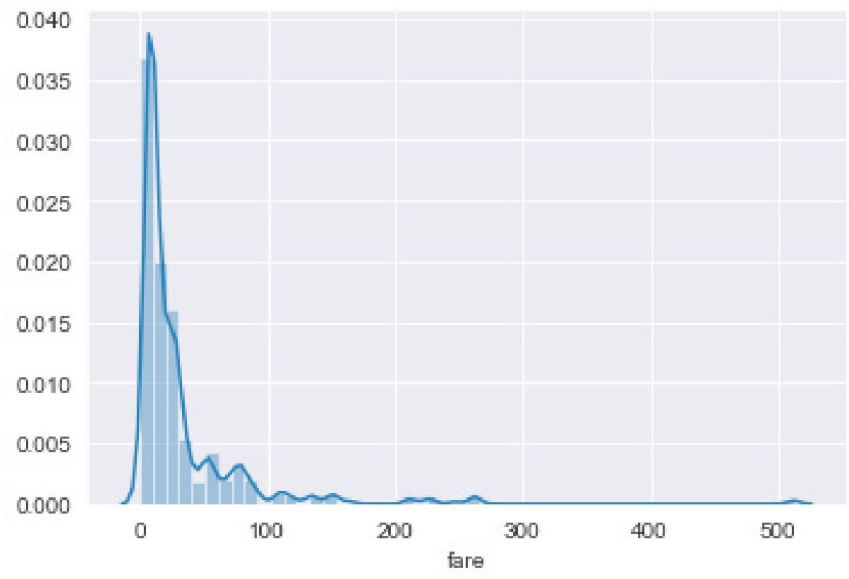
Như đã thảo luận trước đó, dữ liệu số liên tục chứa
thông tin liên tục trong đó dữ liệu có thể chứa phân số.
Trong tập dữ liệu Titanic, cột giá vé chứa liên tục
dữ liệu số vì công bằng có thể ở dạng phân số. Trên thực tế, nếu bạn nhìn vào
giá vé mà khách hàng đã trả trong bản ghi đầu tiên, bạn có thể thấy giá trị
giá vé là 7,2500, đây là một số liên tục
giá trị.

Hãy vẽ một biểu đồ phân phối cho cột giá vé để xem
giá vé được phân phối. Thực hiện tập lệnh sau:

Kịch bản 3:

```
sns.distplot(tips_data['fare'], kde = False)
```

Đầu ra:



Kết quả cho thấy phần lớn hành khách đã trả tiền
giá vé từ 0–20 đô la.

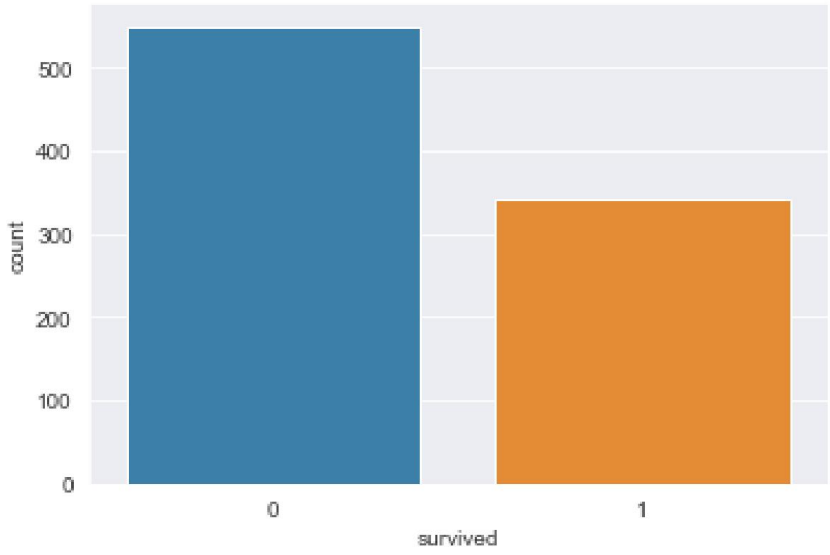
2.2.3. Dữ liệu nhị phân

Có một loại dữ liệu số nữa được gọi là dữ liệu số nhị phân. Dữ liệu số nhị phân, như tên gọi của nó, chỉ có thể có một trong hai giá trị có thể, tức là 0 hoặc 1. Trong tập dữ liệu Titanic, cột survive chứa dữ liệu số nhị phân. Cột survive cho biết hành khách có sống sót sau vụ tai nạn Titanic hay không. Hãy vẽ biểu đồ tần suất đếm những hành khách sống sót và những người không sống sót. Thực hiện đoạn mã sau:

Kịch bản 4:

```
sns.countplot(x='sống sót', dữ liệu=titanic_data)
```

Đầu ra:



Kết quả cho thấy trong số khoảng 900 hành khách, có khoảng 350 người sống sót, trong khi khoảng 550 người không thể sống sót vì thanh
Biểu đồ cho số 0 cho thấy giá trị khoảng 550.

2.3 Phân loại Da

Dữ liệu phân loại là loại dữ liệu trong đó các giá trị được chọn từ một tập hợp được xác định trước. Ví dụ, giới tính của một người là một biến phân loại vì nó chỉ có thể có một trong hai các giá trị được xác định trước, tức là nam và nữ (hoặc trung tính về giới tính). Tương tự như vậy, một ví dụ khác của biến phân loại là quốc tịch của một người, hoặc tên của công ty đó sản xuất ô tô, v.v.

Dữ liệu theo danh mục có thể được chia thành hai loại:

1. Dữ liệu thứ tự
2. Dữ liệu danh nghĩa

2.3.1. Dữ liệu thứ tự

Dữ liệu thứ tự là một loại dữ liệu phân loại trong đó các giá trị được sắp xếp có ý nghĩa và có một số mối quan hệ giữa chúng. Ví dụ, nếu biến chiều cao có thể có ba giá trị, tức là ngắn, trung bình và cao, chúng ta có thể nói rằng chiều cao

biến có dữ liệu thứ tự vì short ngắn hơn medium

và trung bình thì ngắn hơn cao. Nói cách khác, các giá trị theo một thứ tự nhất định. Tương tự, điểm của một học sinh trong một kỳ thi cụ thể cũ ng là một biến thứ tự nếu điểm có thể là A, B, C, D vì điểm A tốt hơn điểm B và điểm B tốt hơn điểm C. Có một thứ tự nhất định giữa các giá trị.

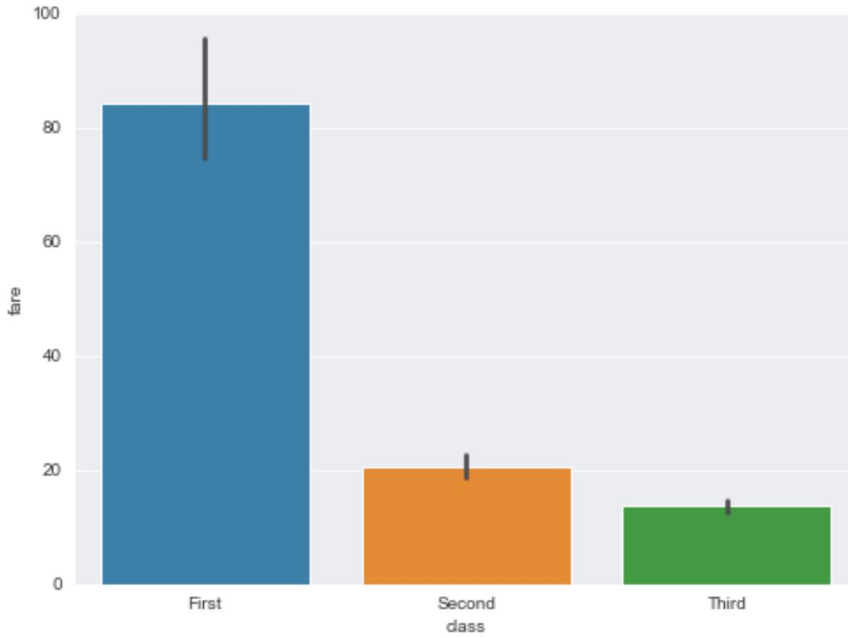
Nếu bạn nhìn vào tập dữ liệu Titanic, bạn có thể thấy rằng lớp
cột chứa dữ liệu thứ tự. Cột lớp có thể có ba

các giá trị có thể, tức là, Hạng nhất, Hạng nhì và Hạng ba. Ở đây có một mối quan hệ nhất định giữa ba giá trị. Hạng nhất đắt hơn hạng hai và hạng hai đắt hơn hạng ba. Chúng ta có thể xác minh điều này bằng cách vẽ biểu đồ giá vé trung bình mà hành khách ở mỗi hạng phải trả.

Kịch bản 5:

```
sns.barplot(x='class', y='fare', data=titanic_data)
```

Đầu ra:



Bạn có thể thấy rằng giá vé trung bình mà hành khách hạng nhất phải trả là khoảng 85, trong khi giá vé trung bình của hành khách hạng hai và hạng ba lần lượt là 20 và 16.

2.3.2. Dữ liệu danh nghĩa

Dữ liệu danh nghĩa là một loại dữ liệu mà các giá trị không có thứ tự có ý nghĩa. Ví dụ, quốc tịch của một người hoàn toàn không liên quan đến quốc tịch của người khác. Nếu bạn xem tập dữ liệu Titanic, cột embark_town chứa

dữ liệu danh nghĩa vì thị trấn của sự lên tàu khác nhau

hành khách không có bất kỳ mối quan hệ nào giữa họ.

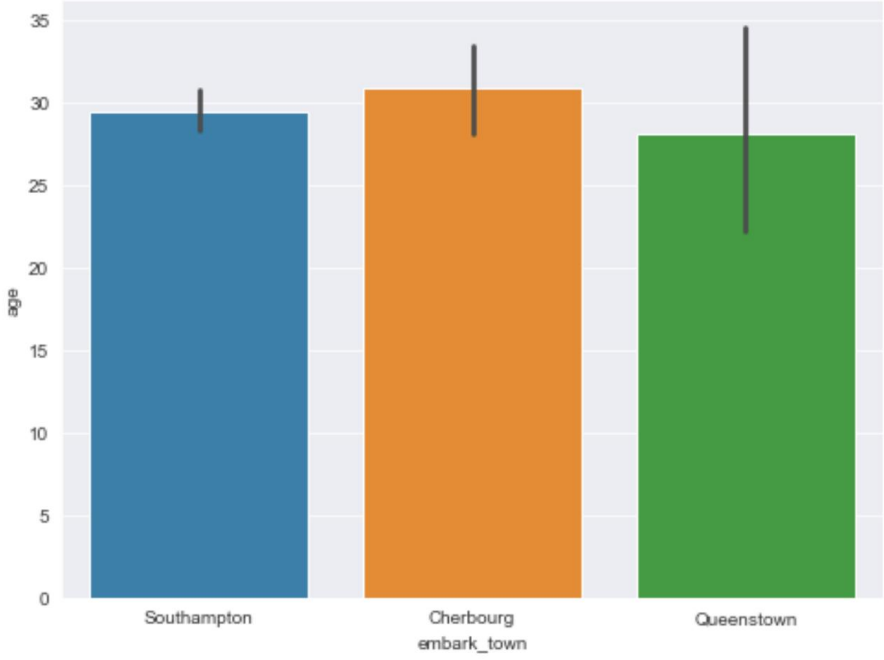
Hãy vẽ biểu đồ trung bình của hành khách lên tàu từ

những thị trấn khác nhau.

Kịch bản 6:

```
sns.barplot(x='embark_town', y='tuổi', dữ liệu=titanic_data)
```

Đầu ra:



Kết quả cho thấy độ tuổi trung bình của hành khách
những người lên tàu từ các thị trấn khác nhau có độ tuổi từ 25–35.

Điều quan trọng cần đề cập là, trong một số trường hợp, các biến phân loại được mã hóa dưới dạng số. Ví dụ,
cột của tập dữ liệu Titanic chứa thông tin về
hành khách có sống sót hay không. Cột sống sót về mặt logic là một biến phân loại
danh nghĩa vì nó chỉ có thể có một trong hai giá trị. Tuy nhiên, các giá trị trong
cột phân loại đã được mã hóa thành 1 và 0. Tương tự như vậy, đôi khi bạn sẽ thấy rằng
các ngày trong tuần được mã hóa bằng số

1 đến 7.

Đọc thêm – Trực quan hóa dữ liệu bằng Python

Để tìm hiểu thêm về Trực quan hóa dữ liệu bằng Python, hãy xem cuốn sách toàn diện của chúng tôi về Trực quan hóa dữ liệu bằng Python (https://amzn.to/39MFgV3)

2.4 Ngày và giờ

Biến ngày và giờ có thể bao gồm ngày và giờ, ngày chỉ, và chỉ thời gian. Sau đây là một số ví dụ về các biến ngày và giờ.

1. Ngày sinh của một người (ví dụ: 10-12-1995, tháng 12 10, 1995, ngày 10 tháng 12 năm 1995).
2. Ngày và giờ của trận đấu bóng đá (ví dụ: 11:30 sáng, Ngày 10 tháng 12 năm 2021).
3. Thời gian văn phòng đóng cửa (ví dụ: 5:00 chiều).

Điều quan trọng cần đề cập là các biến ngày và giờ có thể có nhiều định dạng khác nhau, do đó, một chương hoàn chỉnh đã được dành riêng để tìm hiểu về dữ liệu kiểu ngày và giờ.

2.5 Dữ liệu hỗn hợp

Biến kiểu dữ liệu hỗn hợp là những biến có dữ liệu số và dữ liệu phân loại trong các quan sát khác nhau hoặc chứa dữ liệu số và dữ liệu phân loại trong một quan sát duy nhất.
quan sát một cột dữ liệu.

Ví dụ, nếu có một cột hiển thị số của năm một người đã kết hôn. Cột có thể có các giá trị số như 10, 12, 15 và nó cũ ng có thể có các giá trị chẳng hạn như chưa từng kết hôn, đã ly hôn, v.v. Vì vậy, một cột ở đây chứa cả dữ liệu số cũ ng như dữ liệu phân loại, nhưng một

quan sát ở đây chỉ chứa số hoặc danh mục
dữ liệu.

Mặt khác, nếu có một cột hiển thị
số lượng sản phẩm mà khách hàng đã mua. Cột có thể có các giá trị như 2 kilôgam, 5 gallon, 4 tá, v.v.
Ở đây, biến số lượng có cả dữ liệu số và dữ liệu phân loại trong một quan sát duy nhất.

2.6 Giá trị bị thiếu

Giá trị bị thiếu, như tên gọi của nó, là những quan sát trong tập dữ liệu không chứa bất kỳ giá trị nào. Giá trị bị thiếu có thể thay đổi hoàn toàn các mẫu dữ liệu và do đó, điều cực kỳ quan trọng là phải hiểu lý do tại sao giá trị bị thiếu xảy ra trong tập dữ liệu và cách xử lý chúng. Trong phần này, chúng tôi

sẽ thấy lý do đằng sau sự xuất hiện của các giá trị bị thiếu và những bất lợi của chúng. Trong chương tiếp theo, bạn sẽ thấy các kỹ thuật khác nhau với các ví dụ để xử lý các giá trị bị thiếu bằng ngôn ngữ lập trình Python.

2.6.1. Nguyên nhân của dữ liệu bị thiếu

Có thể có một số lý do khiến giá trị bị thiếu trong tập dữ liệu.
Sau đây là một số ví dụ:

1. Đôi khi dữ liệu được cố ý không được lưu trữ bởi
   con người, dẫn đến việc thiếu các giá trị. Ví dụ,
   hãy xem xét một kịch bản trong đó bạn đang tạo một tập dữ liệu
   thông qua một cuộc khảo sát nơi người dùng nhập dữ liệu. Người dùng có thể nhập hoặc không nhập dữ liệu vào tất cả các trường. Trong những trường hợp như vậy, bạn sẽ không nhận được thông tin cho tất cả các trường trong tập dữ liệu.
2. Một lý do khác đằng sau giá trị bị thiếu có thể là do dữ liệu trong quan sát đang được sử dụng không có sẵn

cho kỹ thuật tính năng. Ví dụ, bạn muốn tạo
một cột hiển thị diện tích của một ngôi nhà. Diện tích có thể
được tính bằng cách nhân các giá trị từ cột chiều dài và chiều rộng. Trong
trường hợp chiều dài hoặc chiều rộng của một quan sát nào đó bị thiếu, cột diện
tích sẽ có giá trị null cho quan sát cụ thể đó.

3. Cuối cùng, các vấn đề tính toán cũ ng có thể dẫn đến việc thiếu giá trị. Ví dụ,
bạn muốn chèn một giá trị vào
cột kết quả từ phép chia hai số. Nếu
mẫu số là 0, kết quả sẽ là vô cực, tức là một giá trị bị thiếu.

2.6.2. Nhược điểm của dữ liệu bị thiếu

Có nhiều bất lợi khi thiếu dữ liệu trong tập dữ liệu của bạn. Chúng như sau:

1. Nhiều thư viện học máy tiên tiến, ví dụ như Scikit
Tìm hiểu không hoạt động với các giá trị bị thiếu trong tập dữ liệu của bạn.
Do đó, các giá trị bị thiếu phải được loại bỏ khỏi
tập dữ liệu, hoặc chúng phải được chuyển đổi thành số
sử dụng kỹ thuật suy đoán dữ liệu bị thiếu.
2. Vấn đề với các kỹ thuật suy diễn dữ liệu bị thiếu là chúng dẫn đến dữ liệu bị bóp
méo vì chúng không thay thế được dữ liệu gốc.

3. Cuối cùng, dữ liệu bị bóp méo có thể ảnh hưởng đến hiệu suất của mô hình thống kê.

2.6.3. Cơ chế đằng sau các giá trị bị thiếu

Trước đó, chúng ta đã thấy lý do chung đằng sau các giá trị bị thiếu. Trong phần này,
bạn sẽ nghiên cứu các cơ chế liên quan đến việc tạo ra các giá trị bị thiếu. Có ba cơ
chế cơ bản:

1. Thiếu dữ liệu hoàn toàn tại Rando

Khi các quan sát bị thiếu không có bất kỳ mối quan hệ nào với bất kỳ cột nào khác trong tập dữ liệu, chúng ta có thể nói rằng dữ liệu đã bị thiếu một cách ngẫu nhiên. Ví dụ, trong một tập dữ liệu, nếu bạn không thể tìm thấy thành phố của một người, thì nó bị thiếu hoàn toàn ngẫu nhiên và bạn không thể tìm ra lý do đằng sau giá trị bị thiếu một cách hợp lý.

2. Thiếu dữ liệu ngẫu nhiên

Nếu các quan sát bị thiếu trong một cột cụ thể có một mối quan hệ với bất kỳ cột nào khác, chúng ta có thể nói rằng dữ liệu là ngẫu nhiên bị thiếu trong các cột này. Ví dụ, phụ nữ có nhiều khả năng không tiết lộ tuổi của mình hơn so với nam giới. Do đó, bạn có thể tìm thấy nhiều giá trị bị thiếu hơn trong độ tuổi cột dành cho phụ nữ, so với nam giới, và do đó, chúng ta có thể nói rằng các giá trị bị thiếu một cách ngẫu nhiên.

3. Dữ liệu bị thiếu không ngẫu nhiên

Trong trường hợp này, bạn có thể gán dữ liệu bị thiếu cho một logic lý do. Ví dụ, nghiên cứu cho thấy bệnh nhân trầm cảm có nhiều khả năng để lại các trường trống trong biểu mẫu so với bệnh nhân không bị trầm cảm. Do đó, dữ liệu bị thiếu ở đây không phải là ngẫu nhiên bị bỏ lỡ. Đã có một thiết lập lý do đằng sau việc thiếu dữ liệu.

2.7 Số lượng trong Da phạm trù

Như đã đề cập trước đó, các cột danh mục chứa các quan sát trong đó các giá trị có thể là bất kỳ tập hợp nhãn nào được xác định trước. Tính số lượng của dữ liệu hoặc một cột dữ liệu đề cập đến số lượng các danh mục hoặc nhãn duy nhất trong cột. Ví dụ, nếu bạn nhìn vào cột giới tính của tập dữ liệu Titanic, nó chỉ có thể có hai giá trị có thể, tức là nam và nữ. Do đó,

số lượng của cột giới tính trong tập dữ liệu Titanic sẽ là 2. Tương tự như vậy, cột lớp có thể có ba giá trị duy nhất.
Do đó, số lượng phần tử của cột lớp là ba.

Một điều cực kỳ quan trọng khác mà bạn nên ghi nhớ là thuật toán thống kê hoạt động với các con số.
Và hầu hết các thư viện máy học tiên tiến, chẳng hạn như Scikit Learn, đều cần dữ liệu về số trong đầu ra.
Do đó, bạn sẽ phải mã hóa dữ liệu phân loại của mình thành số trước khi có thể tạo mô hình thống kê thông qua học máy. Chiều của tập dữ liệu sau khi mã hóa dữ liệu phân loại phụ thuộc vào lược đồ mã hóa. Chúng ta sẽ xem mã hóa biến phân loại chi tiết trong chương sắp tới.

Một điểm quan trọng khác cần cân nhắc khi xử lý số lượng là một số nhãn xuất hiện thường xuyên hơn những nhãn khác. Trong khi đào tạo các mô hình học máy, mô hình đào tạo và kiểm tra phải có sự biểu diễn bằng nhau của tất cả các nhãn trong đầu ra. Trong trường hợp một nhãn chỉ tồn tại trong tập đào tạo, mô hình sẽ bị quá khớp trong giai đoạn đào tạo. Nếu không, nếu nhãn không tồn tại trong tập đào tạo và mô hình bắt gặp các nhãn như vậy trong tập kiểm tra, mô hình sẽ đưa ra dự đoán sai vì nó sẽ không được đào tạo với các nhãn chưa biết trong tập đào tạo. Điều tương tự cũ ng đúng đối với các nhãn hiếm.

2.8 Phân phối xác suất

Một khái niệm cực kỳ quan trọng khác mà bạn nên biết trong khi chuẩn bị dữ liệu của bạn là khái niệm về xác suất phân phối. Phân phối xác suất là một hàm cho chúng ta biết khả năng đạt được các giá trị có thể có của một biến có thể lấy. Ví dụ, khả năng một người có màu đen

tóc lớn hơn tóc xanh. Phân phối xác suất
chức năng được định nghĩa như sau:

Z = p(x)

Trong hàm trên, hàm phân phối xác suất cho chúng ta biết khả năng sau sự kiện, Z sẽ có giá trị cụ thể của X. Tổng xác suất của tất cả các giá trị có thể có của X phải bằng 1.

Có hai loại phân phối xác suất chính:

1. Phân phối đồng đều

Phân phối đồng đều là một loại phân phối mà tất cả các quan sát đều có khả năng xảy ra như nhau. Biểu đồ của phân phối đồng đều là một đường thẳng nằm ngang.

2. Phân phối chuẩn

Mặt khác, phân phối chuẩn, cũ ng là
được gọi là phân phối Gaussian, là một loại phân phối
nơi mà hầu hết các quan sát xảy ra xung quanh trung tâm
đỉnh. Và xác suất cho các giá trị xa đỉnh hơn giảm đều theo cả hai hướng và có khả năng xảy ra như nhau. Phân phối chuẩn thường trông giống như

một chiếc chuông ngược.

Bạn có thể tìm phân phối xác suất bằng biểu đồ histogram.
Hãy tìm phân phối xác suất của tiền boa được trả bởi
khách hàng tại một nhà hàng hư cấu. Đầu tiên, chúng tôi sẽ nhập
Bộ dữ liệu mẹo như được hiển thị bên dưới:

Kịch bản 7:

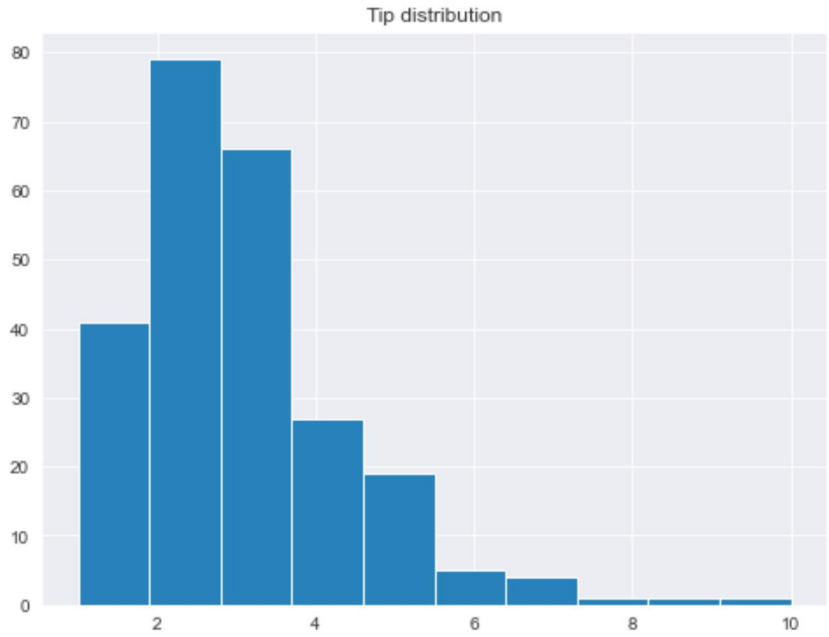
```
tips_data = sns.load_dataset('Mẹo')
mẹo_dữ_liệu.đầu()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Kịch bản 8:

```
plt.title('Phân phối tiền boa')
plt.hist(tips_data[«tip»])
```

Đầu ra:



Đầu ra cho thấy một hình chuông lệch. Điều này được gọi là phân phối lệch dương vì đồ thị được kéo dài đến giá trị dương

bên của giá trị trung bình.

2.9 Giá trị ngoại lệ

Giá trị ngoại lệ là những giá trị quá xa so với phần còn lại
quan sát trong các cột. Ví dụ, nếu cân nặng của hầu hết mọi người trong một mẫu dao động
trong khoảng 50–100 kg, thì quan sát 500 kg sẽ được coi là giá trị ngoại lệ vì quan sát
như vậy hiếm khi xảy ra.

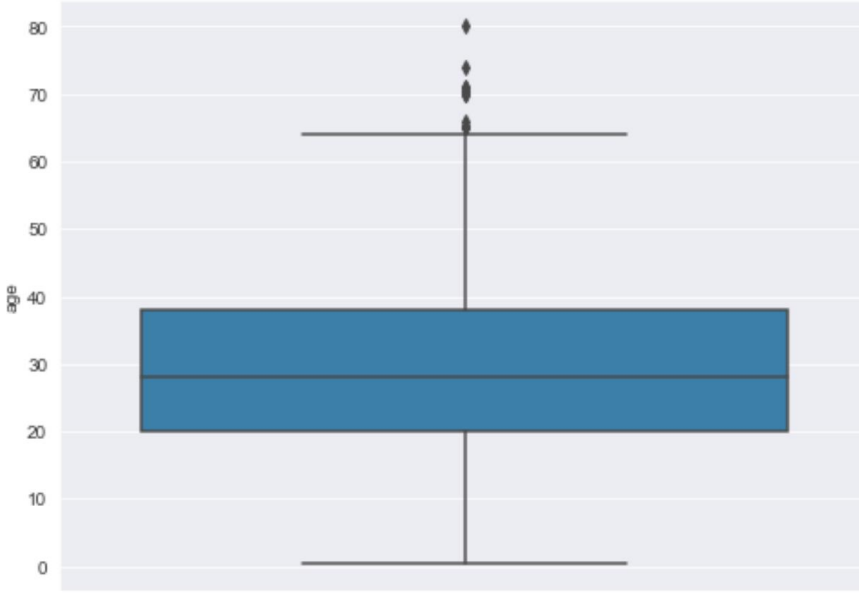
Các giá trị ngoại lệ có thể xảy ra do nhiều lý do khác nhau. Ví dụ, nếu
bạn truy cập tài khoản ngân hàng trực tuyến của mình từ một thành phố cụ thể
95 phần trăm thời gian, kiểm tra trực tuyến vào tài khoản của bạn
từ một thành phố khác sẽ được coi là một ngoại lệ. Một
giá trị ngoại lệ có thể hữu ích vì nó có thể xác định gian lận trực tuyến. Tuy nhiên,
các giá trị ngoại lệ cũ ng có thể xảy ra do lỗi kỹ thuật, lỗi của con người,
khả năng đọc máy, v.v. Trong những trường hợp như vậy, các giá trị ngoại lệ phải là
đã xóa khỏi tập dữ liệu. Chúng tôi sẽ thảo luận về cách xử lý ngoại lệ trong
chi tiết ở chương khác.

Cách tốt nhất để hình dung các giá trị ngoại lai là vẽ biểu đồ hộp. Đoạn mã sau đây vẽ
biểu đồ hộp cho cột tuổi của tập dữ liệu Titanic.

Kịch bản 9:

```
sns.boxplot( y='tuổi', dữ liệu=dữ liệu titanic)
```

Đầu ra:



Biểu đồ hộp Seaborn vẽ thông tin tứ phân vị cùng với các giá trị ngoại lệ. Ở đây, kết quả ở trên cho thấy giá trị trung bình của độ tuổi là khoảng 29 đối với tất cả hành khách trong tập dữ liệu Titanic. Tứ phân vị thứ 4 chứa các giá trị từ 39 đến 65 tuổi. Trên 65 tuổi, bạn có thể thấy các giá trị ngoại lệ dưới dạng các chấm đen. Điều đó có nghĩa là có rất ít hành khách trên 65 tuổi.

Đọc thêm – Thêm về Phân phối Xác suất

Để tìm hiểu thêm về phân phối xác suất, hãy tham khảo tài liệu này. (https://bit.ly/3y7QsaY)

Thời gian thực hành – Bài tập

Bây giờ đến lượt bạn. Thực hiện theo hướng dẫn trong các bài tập bên dưới để kiểm tra hiểu biết cơ bản của bạn về các kiểu dữ liệu. Câu trả lời cho những câu hỏi này được đưa ra ở cuối sách.

Bài tập 2.

Câu hỏi 1:

Kiểu cột đơn lẻ trong tập dữ liệu Titanic là gì?

A. Thứ tự

B. Liên tục

C. Rời rạc

D. Danh nghĩa

Hãy xem tập dữ liệu sau đây. Nó được gọi là tập dữ liệu Mẹo.

Câu hỏi 2–4 dựa trên tập dữ liệu Tips này:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Câu hỏi 2:

Kiểu cột ngày trong tập dữ liệu trên là gì?

A. Thứ tự

B. Liên tục

C. Rời rạc

D. Danh nghĩa

Câu hỏi 3:

Xác định các cột số liên tục trong tập dữ liệu Mẹo.

Câu hỏi 4:

Viết tên các cột rời rạc trong tập dữ liệu Tips:

Câu hỏi 5:

Nên vẽ biểu đồ nào để hình dung xác suất
phân phối trong một tập dữ liệu?

A. Biểu đồ thanh

B. Biểu đồ Histogram

C. Biểu đồ hộp

D. Biểu đồ đường thẳng

Câu hỏi 6:

Sự xuất hiện nhãn hiếm trong một biến phân loại có thể gây ra
_____ trong tập huấn luyện.

A. Dự đoán sai

B. Quá khớp

C. Không vừa vặn

D. Không có câu nào ở trên

Xử lý dữ liệu bị thiếu

3.1 Giới thiệu

Trong các chương trước, bạn đã được giới thiệu về nhiều khái niệm cấp cao mà chúng ta sẽ nghiên cứu trong cuốn sách này. Một trong những khái niệm đó là giá trị bị thiếu. Bạn đã nghiên cứu giá trị bị thiếu là gì, cách giá trị bị thiếu được đưa vào tập dữ liệu và cách chúng ảnh hưởng đến các mô hình thống kê. Trong chương này, bạn sẽ thấy cách xử lý cơ bản các giá trị bị thiếu.

3.2 Phân tích trường hợp hoàn chỉnh

Phân tích trường hợp hoàn chỉnh (CCA), còn được gọi là xóa theo danh sách, là kỹ thuật cơ bản nhất để xử lý dữ liệu bị thiếu. Trong CCA, bạn chỉ cần di chuyển tất cả các hàng hoặc bản ghi mà bất kỳ cột hoặc trường nào chứa giá trị bị thiếu. Chỉ những bản ghi có giá trị thực tế hiện diện cho tất cả các cột trong tập dữ liệu mới được xử lý. CCA có thể được áp dụng để xử lý cả giá trị bị thiếu theo số và theo danh mục.

Bảng sau đây chứa hồ sơ hư cấu về bệnh nhân trong bệnh viện.

Tên	Tuổi	Giới tính	Nhóm máu
Jon	25	Nam giới	Có +
James		Nam giới	Điểm A+
Mike	62	Nam giới	
Nick	42	Nam giới	B-
Harry	45	Nam giới	AB+
Sally	26	Nữ giới	
Laura	35	Nữ giới	B+

Trong bảng trên, độ tuổi của bệnh nhân James bị thiếu, trong khi nhóm máu của bệnh nhân Mike và Sally bị thiếu. Nếu chúng ta sử dụng phương pháp CCA để xử lý các giá trị bị thiếu này, chúng ta sẽ chỉ cần xóa các bản ghi có giá trị bị thiếu và chúng ta sẽ có tập dữ liệu sau:

Tên	Tuổi	Giới tính	Nhóm máu
Jon	25	Nam giới	Có +
Nick	42	Nam giới	B-
Harry	45	Nam giới	AB+
Laura	35	Nữ giới	B+

Ưu điểm của CCA

Giả định đằng sau CCA là dữ liệu bị thiếu một cách ngẫu nhiên.
CCA cực kỳ dễ áp dụng và không liên quan đến kỹ thuật thống kê. Cuối cùng, phân phối của các biến cũ ng được bảo toàn.

Nhược điểm của CCA

Nhược điểm chính của CCA là nếu một tập dữ liệu chứa một số lượng lớn các giá trị bị thiếu, một tập hợp con lớn dữ liệu sẽ bị CCA loại bỏ. Ngoài ra, nếu các giá trị không bị thiếu một cách ngẫu nhiên, CCA có thể tạo ra một tập dữ liệu bị thiên vị. Cuối cùng, các mô hình thống kê được đào tạo trên một tập dữ liệu mà CCA được áp dụng không có khả năng xử lý các giá trị bị thiếu trong quá trình sản xuất.

Theo nguyên tắc chung, nếu bạn chắc chắn rằng các giá trị bị thiếu hoàn toàn ngẫu nhiên và tỷ lệ hồ sơ bị thiếu giá trị nhỏ hơn 5 phần trăm, bạn có thể sử dụng CAA để xử lý những giá trị đó giá trị bị thiếu.

Trong các phần tiếp theo, chúng ta sẽ xem cách xử lý dữ liệu số và dữ liệu danh mục bị thiếu.

3.3 Xử lý số liệu bị thiếu

Ở chương trước, bạn đã nghiên cứu các loại dữ liệu khác nhau mà bạn sẽ gặp phải trong sự nghiệp khoa học dữ liệu của mình.
Một trong những kiểu dữ liệu phổ biến nhất là dữ liệu số, bao gồm các con số. Để xử lý dữ liệu số bị thiếu, chúng ta có thể sử dụng các kỹ thuật thống kê. Việc sử dụng các kỹ thuật thống kê hoặc thuật toán để thay thế các giá trị bị thiếu bằng các giá trị được tạo ra theo thống kê được gọi là imputation.

3.3.1. Quy kết trung bình hoặc trung vị

Imputation trung bình hoặc trung vị là một trong những kỹ thuật imputation được sử dụng phổ biến nhất để xử lý dữ liệu số bị thiếu. Trong imputation trung bình hoặc trung vị, các giá trị bị thiếu trong một cột được thay thế bằng giá trị trung bình hoặc trung vị của tất cả các giá trị còn lại trong cột cụ thể đó.

Ví dụ, nếu bạn có một cột có dữ liệu sau:

Tuổi
15
Không có
20
25
40

Trong cột Tuổi ở trên, giá trị thứ hai bị thiếu.

Do đó, với phép quy ước trung bình và trung vị, bạn có thể thay thế giá trị thứ hai có giá trị trung bình hoặc trung vị của tất cả các giá trị khác trong cột. Ví dụ, cột sau chứa giá trị trung bình của tất cả các giá trị còn lại, tức là 25 ở hàng thứ hai. Bạn cũ ng có thể thay thế giá trị này bằng giá trị trung vị nếu muốn.

Tuổi
15
25
20
25
40

Chúng ta hãy xem một ví dụ thực tế về cách tính giá trị trung bình và trung vị. Chúng tôi sẽ nhập tập dữ liệu Titanic và tìm các cột chứa các giá trị bị thiếu. Sau đó, chúng ta sẽ áp dụng phép tính trung bình và trung vị cho các cột chứa các giá trị bị thiếu và cuối cùng, chúng ta sẽ thấy hiệu ứng của việc áp dụng phép tính trung bình và trung vị cho các giá trị bị thiếu.

Bạn không cần phải tải xuống tập dữ liệu Titanic. Nếu bạn nhập thư viện Seaborn, dữ liệu Titanic sẽ được tải xuống cùng với nó. Tập lệnh sau đây nhập tập dữ liệu Titanic và hiển thị
năm hàng đầu tiên.

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 một số cột số từ tập dữ liệu và xem liệu chúng có chứa giá trị bị thiếu hay không.

Kịch bản 2:

```
titanic_data = titanic_data[["sống sót", "pclass", "tuổi", "giá vé"]]

titanic_data.đầu()
```

Đầu ra:

	survived	pclass	age	fare
0	0	3	22.0	7.2500
1	1	1	38.0	71.2833
2	1	3	26.0	7.9250
3	1	1	35.0	53.1000
4	0	3	35.0	8.0500

Để tìm các giá trị còn thiếu từ các cột đã đề cập ở trên, trước tiên bạn cần gọi phương thức isnull() trên titanic_
data dataframe, và sau đó bạn cần gọi phương thức mean()
như được hiển thị bên dưới:

Kịch bản 3:

```
titanic_data.isnull().mean()
```

Đầu ra:

```
sống sót 0.000000
lớp p 0.000000
tuổi 0,198653
giá vé 0,000000
Kiểu dữ liệu: float64
```

Kết quả cho thấy chỉ có cột tuổi chứa các giá trị bị thiếu. Và tỷ lệ các giá trị bị thiếu là khoảng 19,86 phần trăm.

Bây giờ chúng ta hãy tìm giá trị trung vị và giá trị trung bình cho tất cả các giá trị không giá trị bị thiếu trong cột tuổi .

Kịch bản 4:

```
trung vị = titanic_data.age.median()
in(trung vị)

trung bình = titanic_data.age.mean()
in(trung bình)
```

Đầu ra:

```
28.0
29.69911764705882
```

Cột độ tuổi có giá trị trung bình là 28 và giá trị trung vị là 29,6991.

Để vẽ biểu đồ mật độ hạt nhân cho độ tuổi thực tế, độ tuổi trung vị và độ tuổi trung bình, chúng ta sẽ thêm các cột vào khung dữ liệu Pandas.

Kịch bản 5:

```
nhập numpy dưới dạng np

titanic_data['Tuổi trung bình'] = titanic_data.age.fillna(trung bình)

titanic_data['Tuổi_trung_bình'] = titanic_data.age.fillna(trung_bình)

titanic_data['Tuổi_trung_bình'] = np.round(titanic_data['Tuổi_trung_bình'], 1)


titanic_data.head(20)
```

Đoạn mã trên thêm các cột Median_Age và Mean_Age vào khung dữ liệu titanic_data và in ra 20 bản ghi đầu tiên.

Sau đây là kết quả của đoạn mã trên:

Đầu ra:

	survived	pclass	age	fare	Median_Age	Mean_Age
0	0	3	22.0	7.2500	22.0	22.0
1	1	1	38.0	71.2833	38.0	38.0
2	1	3	26.0	7.9250	26.0	26.0
3	1	1	35.0	53.1000	35.0	35.0
4	0	3	35.0	8.0500	35.0	35.0
5	0	3	NaN	8.4583	28.0	29.7
6	0	1	54.0	51.8625	54.0	54.0
7	0	3	2.0	21.0750	2.0	2.0
8	1	3	27.0	11.1333	27.0	27.0
9	1	2	14.0	30.0708	14.0	14.0
10	1	3	4.0	16.7000	4.0	4.0
11	1	1	58.0	26.5500	58.0	58.0
12	0	3	20.0	8.0500	20.0	20.0
13	0	3	39.0	31.2750	39.0	39.0
14	0	3	14.0	7.8542	14.0	14.0
15	1	2	55.0	16.0000	55.0	55.0
16	0	3	2.0	29.1250	2.0	2.0
17	1	2	NaN	13.0000	28.0	29.7
18	0	3	31.0	18.0000	31.0	31.0
19	1	3	NaN	7.2250	28.0	29.7

Các hàng được tô sáng trong kết quả đầu ra ở trên cho thấy NaN, tức là giá trị null trong cột tuổi, đã được thay thế bằng giá trị trung vị trong cột Median_Age và bằng giá trị trung bình trong cột Mean_Age.

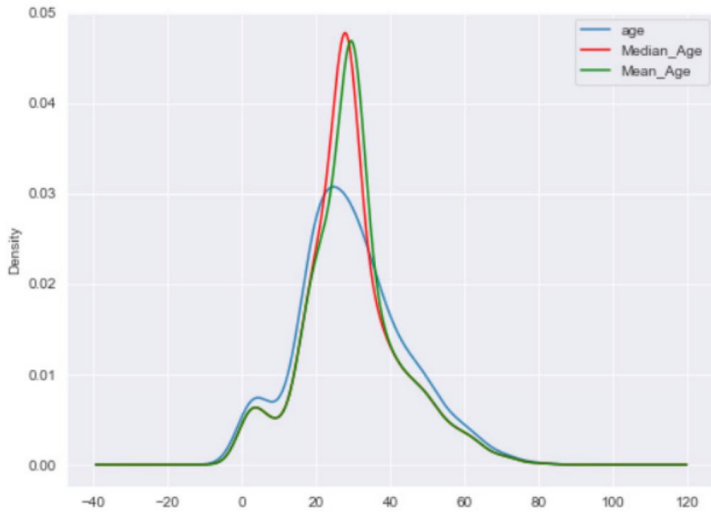
Việc ước tính giá trị trung bình và trung vị có thể ảnh hưởng đến phân phối dữ liệu cho các cột chứa các giá trị bị thiếu.

Cụ thể, phương sai của cột được giảm đi nhờ phép tính trung bình và trung vị vì giờ đây có nhiều giá trị hơn được thêm vào tâm của phân phối. Đoạn mã sau đây vẽ biểu đồ phân phối dữ liệu cho độ tuổi, Median_Age và Mean_Age cột.

Kịch bản 6:

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

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

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

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


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

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

Sau đây là kết quả của đoạn mã trên:



Bạn có thể thấy rõ ràng rằng các giá trị mặc định trong các cột tuổi đã bị bóp méo bởi phép quy ước trung bình và trung vị, và
độ biến thiên tổng thể của tập dữ liệu cũ ng đã giảm.

Khuyến nghị

Có thể sử dụng phép tính trung bình và trung vị cho dữ liệu số bị thiếu trong trường hợp dữ liệu bị thiếu ngẫu nhiên. Nếu dữ liệu phân phối chuẩn, phép tính trung bình sẽ tốt hơn, hoặc phép tính trung vị được ưu tiên trong trường hợp phân phối lệch.

Thuận lợi

Việc quy ước giá trị trung bình và trung vị dễ thực hiện và là một chiến lược hữu ích để nhanh chóng có được một tập dữ liệu lớn. Hơn nữa, việc quy ước giá trị trung bình và trung vị có thể được thực hiện trong giai đoạn sản xuất.

Nhược điểm

Như đã nói ở trên, nhược điểm lớn nhất của việc quy ước giá trị trung bình và trung vị là nó ảnh hưởng đến phân phối dữ liệu mặc định và
phương sai và hiệp phương sai của dữ liệu.

3.3.2. Kết thúc phân phối

Phép tính trung bình và trung vị và CCA không phải là những kỹ thuật tốt để tính giá trị bị thiếu trong trường hợp dữ liệu không bị thiếu ngẫu nhiên. Đối với dữ liệu bị thiếu ngẫu nhiên, các kỹ thuật thường được sử dụng nhất là phép tính kết thúc phân phối/kết thúc đuôi. Trong phép tính kết thúc đuôi, một giá trị được chọn từ phần đuôi của dữ liệu. Giá trị này biểu thị rằng dữ liệu thực tế cho bản ghi đã bị thiếu. Do đó, dữ liệu không bị thiếu ngẫu nhiên có thể được tính đến trong khi đào tạo

mô hình thống kê trên dữ liệu.

Trong trường hợp dữ liệu phân phối chuẩn, giá trị cuối phân phối có thể được tính bằng cách nhân giá trị trung bình với ba độ lệch chuẩn. Trong trường hợp phân phối dữ liệu lệch, có thể sử dụng Quy tắc liên tứ phân vị để tìm giá trị đuôi.

IQR = Phân vị thứ 75 – Phân vị thứ 25

Giới hạn IQR trên = Phân vị thứ 75 + IQR x 1,5

Giới hạn IQR thấp hơn = Phân vị thứ 25 – IQR x 1,5

Chúng ta hãy thực hiện phép tính cuối cùng của phép tính đuôi trên cột tuổi của bộ dữ liệu Titanic.

Đoạn mã sau đây nhập tập dữ liệu Titanic, lọc các cột số và sau đó tìm phần trăm bị thiếu
giá trị trong mỗi cột.

Kịch bản 7:

```
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[["sống sót", "pclass", "tuổi", "giá vé"]]



titanic_data.isnull().mean()
```

Đầu ra:

```
sống sót 0.000000
lớp p 0.000000
tuổi 0,198653
giá vé 0,000000
Kiểu dữ liệu: float64
```

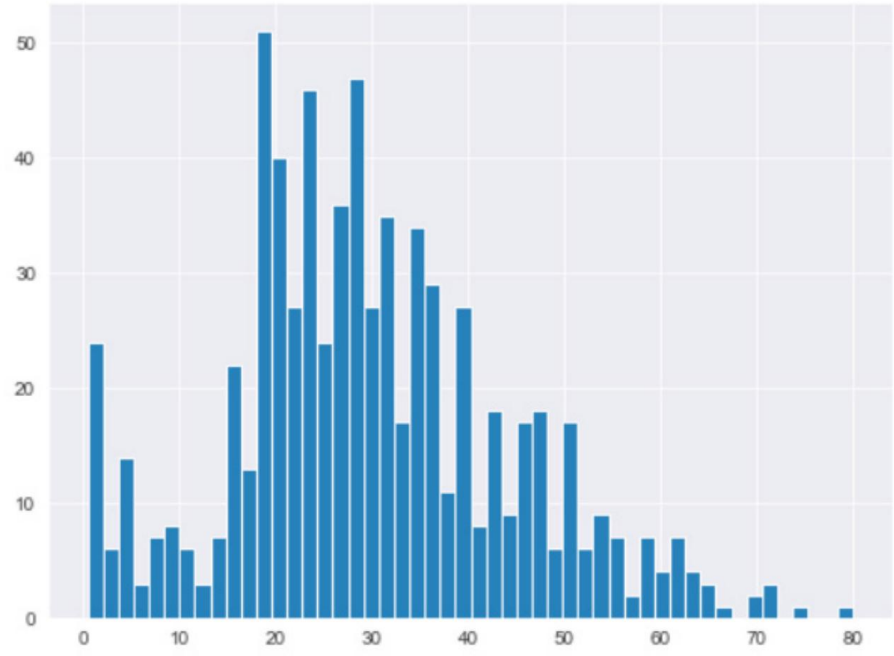
Kết quả đầu ra ở trên cho thấy chỉ có cột tuổi bị thiếu giá trị, chiếm khoảng 20 phần trăm toàn bộ tập dữ liệu.

Bước tiếp theo là vẽ biểu đồ phân phối dữ liệu cho cột tuổi . Biểu đồ histogram có thể tiết lộ phân phối dữ liệu của cột.

Kịch bản 8:

```
titanic_data.age.hist(thùng=50)
```

Đầu ra:



Đầu ra cho thấy cột tuổi có phân phối gần như chuẩn. Do đó, giá trị cuối của phân phối có thể được tính bằng cách nhân giá trị trung bình của cột tuổi với ba độ lệch chuẩn.

Kết quả đầu ra ở trên một lần nữa cho thấy rằng,

Kịch bản 9:

```
eod_value = titanic_data.age.mean() + 3 * titanic_data.age.
tiêu chuẩn()
in(giá_trị_eod)
```

Đầu ra:

73.278

Cuối cùng, các giá trị bị thiếu trong cột tuổi có thể được thay thế bằng giá trị cuối được tính toán trong tập lệnh 9.

Kịch bản 10:

```
nhập numpy dưới dạng np

titanic_data['age_eod'] = titanic_data.age.fillna(eod_value)
titanic_data.head(20)
```

Đầu ra:

	survived	pclass	age	fare	age_eod
0	0	3	22.0	7.2500	22.00000
1	1	1	38.0	71.2833	38.00000
2	1	3	26.0	7.9250	26.00000
3	1	1	35.0	53.1000	35.00000
4	0	3	35.0	8.0500	35.00000
5	0	3	NaN	8.4583	73.27861
6	0	1	54.0	51.8625	54.00000
7	0	3	2.0	21.0750	2.00000
8	1	3	27.0	11.1333	27.00000
9	1	2	14.0	30.0708	14.00000
10	1	3	4.0	16.7000	4.00000
11	1	1	58.0	26.5500	58.00000
12	0	3	20.0	8.0500	20.00000
13	0	3	39.0	31.2750	39.00000
14	0	3	14.0	7.8542	14.00000
15	1	2	55.0	16.0000	55.00000
16	0	3	2.0	29.1250	2.00000
17	1	2	NaN	13.0000	73.27861
18	0	3	31.0	18.0000	31.00000
19	1	3	NaN	7.2250	73.27861

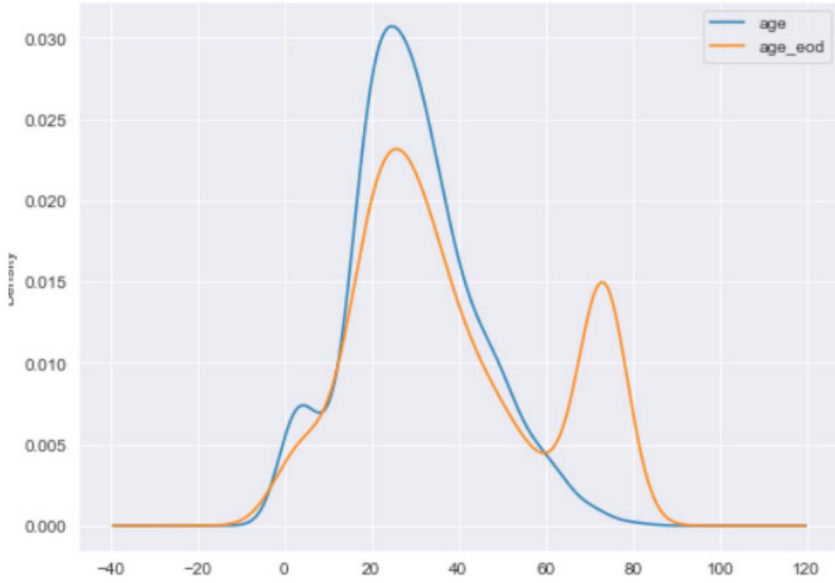
Đầu ra ở trên cho thấy giá trị cuối phân phối, tức là ~73, đã thay thế các giá trị NaN trong cột tuổi.

Cuối cùng, bạn có thể vẽ biểu đồ ước tính mật độ hạt nhân cho cột tuổi ban đầu và cột tuổi với phép suy đoán kết thúc phân phối.

Kịch bản 11:

```
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_eod'] .plot(loại='kde', ax=ax)
dòng, nhãn = ax.get_legend_handles_labels()
ax.legend(dòng, nhãn, loc='tốt nhất')
```

Đầu ra:



Ưu điểm và nhược điểm

Một trong những lợi thế chính của việc tính toán cuối phân phối là nó có thể được áp dụng cho tập dữ liệu có giá trị
không bị thiếu một cách ngẫu nhiên. Những lợi thế khác của việc quy kết cuối phân phối bao gồm tính đơn giản để hiểu, khả năng tạo ra các tập dữ liệu lớn trong thời gian ngắn và khả năng áp dụng trong môi trường sản xuất.

Những nhược điểm bao gồm sự bóp méo phân phối dữ liệu, phương sai và hiệp phương sai.

3.3.3. Quy kết giá trị tùy ý

Cuối cùng, khi phân bổ giá trị, các giá trị thay thế các giá trị bị thiếu sẽ được tính toán từ dữ liệu, trong khi khi phân bổ giá trị tùy ý, các giá trị được sử dụng để thay thế các giá trị bị thiếu sẽ được chọn tùy ý.

Các giá trị tùy ý được chọn theo cách mà chúng không thuộc về tập dữ liệu; thay vào đó, chúng biểu thị các giá trị bị thiếu.
Giá trị tốt để chọn là 99, 999 hoặc bất kỳ số nào chứa số 9. Trong trường hợp tập dữ liệu chỉ chứa giá trị dương, có thể chọn 1 làm số tùy ý.

Hãy áp dụng giá trị quy ước tùy ý vào cột tuổi
của tập dữ liệu Titanic.

Tập lệnh sau đây nhập tập dữ liệu Titanic, lọc một số
các cột số và hiển thị phần trăm bị thiếu
giá trị trong các cột đó.

Kịch bản 12:

```
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[["sống sót", "pclass", "tuổi", "giá vé"]]



titanic_data.isnull().mean()
```

Đầu ra:

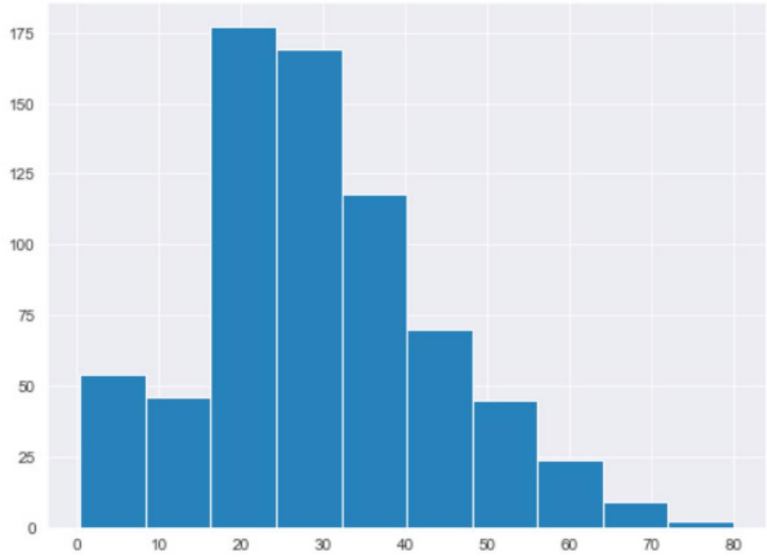
```
sống sót 0.000000
lớp p 0.000000
tuổi 0,198653
giá vé 0.000000
Kiểu dữ liệu: float64
```

Đầu ra cho thấy chỉ có cột tuổi chứa một số giá trị bị thiếu. Tiếp theo, chúng tôi vẽ biểu đồ histogram cho cột tuổi để xem phân phối dữ liệu.

Kịch bản 13:

```
titanic_data.tuổi.hist()
```

Đầu ra:



Đầu ra cho thấy giá trị dương lớn nhất là khoảng 80.
Do đó, 99 có thể là một giá trị tùy ý rất tốt. Hơn nữa, vì cột tuổi chỉ chứa các giá trị dương, 1 có thể là một giá trị tùy ý rất hữu ích khác. Hãy thay thế các giá trị bị thiếu trong cột tuổi trước bằng 99, sau đó bằng 1.

Kịch bản 14:

```
nhập numpy dưới dạng np

titanic_data['tuổi_99'] = titanic_data.age.fillna(99)

titanic_data['age_minus1'] = titanic_data.age.fillna(-1)


titanic_data.head(20)
```

Đầu ra:

	survived	pclass	age	fare	age_99	age_minus1
0	0	3	22.0	7.2500	22.0	22.0
1	1	1	38.0	71.2833	38.0	38.0
2	1	3	26.0	7.9250	26.0	26.0
3	1	1	35.0	53.1000	35.0	35.0
4	0	3	35.0	8.0500	35.0	35.0
5	0	3	NaN	8.4583	99.0	-1.0
6	0	1	54.0	51.8625	54.0	54.0
7	0	3	2.0	21.0750	2.0	2.0
8	1	3	27.0	11.1333	27.0	27.0
9	1	2	14.0	30.0708	14.0	14.0
10	1	3	4.0	16.7000	4.0	4.0
11	1	1	58.0	26.5500	58.0	58.0
12	0	3	20.0	8.0500	20.0	20.0
13	0	3	39.0	31.2750	39.0	39.0
14	0	3	14.0	7.8542	14.0	14.0
15	1	2	55.0	16.0000	55.0	55.0
16	0	3	2.0	29.1250	2.0	2.0
17	1	2	NaN	13.0000	99.0	-1.0
18	0	3	31.0	18.0000	31.0	31.0
19	1	3	NaN	7.2250	99.0	-1.0