AULA-2 CÓDIGOS

terça-feira, 19 de setembro de 2023 23:09

Construindo uma distribuição de frequência com base em um conjunto de dados:

```
import pandas as pd
# Conjunto de dados
data = pd.Series([128, 100, 180, 150, 200, 90, 340, 105, 85, 270,
                  200, 65, 230, 150, 150, 120, 130, 80, 230, 200,
                  110, 126, 170, 132, 140, 112, 90, 340, 170, 190])
print(data)
```

Este código é utilizado para criar um conjunto de dados representado por uma Pandas Series e exibi-lo. Vou explicar cada trecho:

1. `import pandas as pd`: Isso importa a biblioteca Pandas com o alias "pd", que é comumente usado para simplificar o uso da biblioteca.

2. `data = pd.Series([...])`: Aqui, você está criando uma Pandas Series chamada "data". Uma Pandas Series é uma estrutura de dados unidimensional semelhante a uma matriz ou lista, mas com rótulos para cada elemento. No interior dos colchetes `[...]`, você especifica uma lista de valores inteiros que formarão os elementos desta série. Esses valores representam algum conjunto de dados.

3. `print(data)`: Este comando imprime a série "data" na saída padrão, que exibirá os valores armazenados na série.

O conjunto de dados em questão consiste em uma série de valores inteiros, representando possivelmente algum tipo de medição ou contagem, mas o código por si só não fornece informações adicionais sobre o significado desses números.

```
# Valores máximo, mínimo e número de classes
v_max = data.max()
v_min = data.min()
n_classes = 7
print(f'Valor mínimo da distribuição = {v_min}')
print(f'Valor máximo da distribuição = {v_max}')
```

Neste trecho de código, você está realizando algumas operações estatísticas nos dados da série. Vou explicar cada linha:

1. `v_max = data.max()`: Aqui, você está usando o método `max()` da série Pandas "data" para encontrar o valor máximo (maior valor) presente nos dados.

2. `v_min = data.min()`: Similarmente, você está usando o método `min()` para encontrar o valor mínimo (menor valor) presente nos dados.

3. `n_classes = 7`: Neste ponto, você define a variável `n_classes` com o valor 7. Isso parece estar relacionado à criação de classes para a construção de um histograma ou distribuição de frequência.

4. `print(f'Valor mínimo da distribuição = {v_min}')`: Aqui, você imprime na saída padrão o valor mínimo encontrado nos dados.

5. `print(f'Valor máximo da distribuição = {v_max}')`: Da mesma forma, você imprime o valor máximo encontrado nos dados.

Basicamente, este trecho do código é uma preparação para calcular e exibir estatísticas sobre os dados, como os valores mínimo e máximo, que são frequentemente usados na análise de dados e na construção de histogramas ou distribuições de frequência.

Distribuição de Frequências:

```
dist_freq = data.value_counts(bins=7).sort_index()
print(dist_freq)
```

Neste trecho de código, você está criando uma distribuição de frequência dos dados da série "data". Vou explicar cada linha:

1. `dist_freq = data.value_counts(bins=7).sort_index()`: Aqui, você está fazendo o seguinte:

   - `data.value_counts(bins=7)`: O método `value_counts()` é usado para contar a frequência de valores únicos na série "data". A opção `bins=7` indica que você deseja criar 7 intervalos (classes) para a distribuição de frequência. Isso divide os dados em 7 classes com base em seus valores.

   - `.sort_index()`: Em seguida, você usa o método `sort_index()` para classificar as classes da distribuição de frequência em ordem crescente com base nos valores dos intervalos. Isso garante que as classes estejam organizadas de forma adequada.

O resultado disso é que a variável `dist_freq` conterá a distribuição de frequência dos dados, com as classes organizadas em ordem crescente e a contagem de valores em cada classe.

Depois de executar este código, a variável `dist_freq` será uma série Pandas que mostrará a frequência de valores em cada classe da distribuição de frequência, permitindo uma análise mais fácil dos dados.

```
# Reconfiguração do conjunto de dados: Pandas Series para Pandas DataFrame
nova_dist_freq = pd.DataFrame(dist_freq)
display(nova_dist_freq)
```

Neste trecho de código, você está reconfigurando a distribuição de frequência, que inicialmente estava em uma série Pandas, para um DataFrame Pandas. Vou explicar cada linha:

1. `nova_dist_freq = pd.DataFrame(dist_freq)`: Aqui, você está criando um novo DataFrame chamado `nova_dist_freq` a partir da série `dist_freq`. A função `pd.DataFrame()` é usada para criar um DataFrame a partir de diferentes tipos de dados, incluindo séries Pandas.

2. `display(nova_dist_freq)`: Esta linha exibe o novo DataFrame `nova_dist_freq` para que você possa inspecionar a distribuição de frequência agora representada como um DataFrame.

O resultado disso é que a variável `nova_dist_freq` conterá a mesma informação da distribuição de frequência, mas agora organizada em um DataFrame, o que pode ser mais conveniente para análises posteriores ou para apresentar os resultados de maneira mais estruturada.

```
# Resetando os índices do DataFrame
nova_dist_freq = nova_dist_freq.reset_index()
display(nova_dist_freq)
```

1. `nova_dist_freq = nova_dist_freq.reset_index()`: Aqui, você está usando o método `.reset_index()` para redefinir os índices do DataFrame `nova_dist_freq`. Quando você cria uma distribuição de frequência, os valores únicos do conjunto de dados original são usados como índices. Resetar os índices é útil quando você deseja que o DataFrame tenha índices sequenciais numéricos, em vez dos valores originais do conjunto de dados.

2. `display(nova_dist_freq)`: Esta linha exibe o DataFrame `nova_dist_freq` após a redefinição dos índices para que você possa ver como os índices foram alterados.

O resultado disso é que o DataFrame `nova_dist_freq` agora terá índices numéricos sequenciais, o que pode ser mais conveniente para algumas operações de manipulação de dados e visualização. Os valores originais que eram os índices anteriores agora foram movidos para uma coluna regular de dados no DataFrame.

```
# Renomeando as colunas do DataFrame
nova_dist_freq.columns = ['Classe', 'Frequência']
display(nova_dist_freq)
```

Neste trecho de código, você está renomeando as colunas do DataFrame `nova_dist_freq`. Vou explicar cada linha:

1. `nova_dist_freq.columns = ['Classe', 'Frequência']`: Aqui, você está atribuindo uma nova lista de nomes de colunas ao atributo `columns` do DataFrame `nova_dist_freq`. Isso efetivamente renomeia as colunas do DataFrame. No seu caso, você está renomeando as colunas para 'Classe' e 'Frequência', que são nomes mais descritivos para as informações que elas contêm.

2. `display(nova_dist_freq)`: Esta linha exibe o DataFrame `nova_dist_freq` após a renomeação das colunas, para que você possa ver os nomes das colunas atualizados.

O resultado disso é que o DataFrame `nova_dist_freq` agora tem colunas com os nomes 'Classe' e 'Frequência', tornando-o mais legível e descritivo em relação aos dados de frequência que ele contém.

```
# Cálculo da amplitude de classe, com respectivos intervalos de classe
amplitude = round((v_max-v_min)/n_classes)
intervalos = pd.interval_range(start=v_min, end=v_max, freq=amplitude)
print(intervalos)
```

Neste trecho de código, você está calculando a amplitude de classe e criando intervalos de classe para organizar seus dados em grupos. Aqui está a explicação de cada linha:

1. `amplitude = round((v_max-v_min)/n_classes)`: Você está calculando a amplitude de classe, que é a largura de cada intervalo que será usado para agrupar os dados. Isso é feito subtraindo o valor mínimo (`v_min`) do valor máximo (`v_max`) dos dados e dividindo pelo número de classes desejado (`n_classes`). A função `round` é usada para arredondar o resultado para um número inteiro, pois a amplitude de classe deve ser um número inteiro.

2. `intervalos = pd.interval_range(start=v_min, end=v_max, freq=amplitude)`: Aqui, você está criando os intervalos de classe usando a função `pd.interval_range` do Pandas. Você especifica o ponto de partida (`start`) como o valor mínimo (`v_min`) dos seus dados, o ponto final (`end`) como

o valor máximo (`v_max`) dos seus dados e a frequência (`freq`) como a amplitude de classe calculada anteriormente. Isso cria uma série de intervalos igualmente espaçados que serão usados para agrupar seus dados.

3. `print(intervalos)`: Esta linha imprime os intervalos de classe criados. Você verá uma lista de intervalos que representam os grupos nos quais seus dados serão agrupados.

Esses intervalos de classe são úteis para construir um histograma ou calcular frequências acumuladas, facilitando a análise e a visualização de dados agrupados.

```
# Atribuição dos intervalos para a coluna Classe
nova_dist_freq['Classe'] = intervalos
display(nova_dist_freq)
```

Neste trecho de código, você está atribuindo os intervalos de classe calculados anteriormente à coluna "Classe" do seu DataFrame "nova_dist_freq". Aqui está a explicação dessa linha:

1. `nova_dist_freq['Classe'] = intervalos`: Você está usando a coluna "Classe" do DataFrame "nova_dist_freq" como alvo e a está preenchendo com os intervalos de classe que foram calculados. Isso associa cada intervalo de classe ao seu respectivo grupo de frequência no DataFrame.

Essa etapa é importante, pois agora você tem seus dados organizados em intervalos de classe, facilitando a análise e a representação gráfica, como a construção de um histograma. A coluna "Classe" contém esses intervalos para cada classe de frequência.

Características adicionais da distribuição:

```
# Cálculo dos pontos médios das classe
pts_medios = [inter.mid for inter in intervalos]
print(pts_medios)
```

Neste trecho de código, você está calculando os pontos médios de cada intervalo de classe e armazenando esses valores na lista "pts_medios". Aqui está a explicação dessa linha:

1. `pts_medios = [inter.mid for inter in intervalos]`: Você está criando uma lista chamada "pts_medios" usando uma compreensão de lista. Para cada intervalo de classe (representado por "inter") no objeto de intervalos que você calculou anteriormente, o código está calculando o ponto médio desse intervalo com a função `.mid`. Os valores resultantes dos pontos médios são armazenados na lista "pts_medios".

Essa etapa é importante, pois os pontos médios são usados posteriormente em análises estatísticas e na construção de um histograma. Eles representam o valor central de cada intervalo de classe e são úteis para visualizar a distribuição dos dados.

```
# Inserção da coluna Pontos médios
nova_dist_freq['Pontos médios'] = pts_medios
display(nova_dist_freq)
```

Neste trecho de código, você está adicionando uma nova coluna chamada "Pontos médios" ao DataFrame "nova_dist_freq". Aqui está a explicação dessa linha:

1. `nova_dist_freq['Pontos médios'] = pts_medios`: Você está atribuindo a lista de pontos médios (calculados anteriormente) à nova coluna "Pontos médios" no DataFrame "nova_dist_freq". Isso significa que cada linha do DataFrame terá um valor correspondente na coluna "Pontos médios" que representa o ponto médio do intervalo de classe associado a essa linha.

Essa adição da coluna de pontos médios é útil para análises estatísticas posteriores, construção de gráficos e representação visual dos dados em um histograma, pois os pontos médios fornecem uma medida representativa de cada intervalo de classe.

```
# Cálculo e inserção das colunas de frequência relativa e acumulada
observacoes = len(data)
nova_dist_freq['Frequência relativa'] = nova_dist_freq['Frequência']/observacoes
nova_dist_freq['Frequência acumulada'] = nova_dist_freq['Frequência'].cumsum()
display(nova_dist_freq)
```

Neste trecho de código, você está calculando e inserindo duas novas colunas, "Frequência relativa" e "Frequência acumulada", no DataFrame "nova_dist_freq". Aqui está a explicação de cada linha:

1. `observacoes = len(data)`: Aqui, você está contando o número total de observações em seu conjunto de dados "data" usando a função "len". Isso representa o tamanho total do conjunto de dados.

2. `nova_dist_freq['Frequência relativa'] = nova_dist_freq['Frequência']/observacoes`: Nesta linha, você está calculando a frequência relativa para cada classe. A frequência relativa é calculada dividindo a frequência absoluta (número de observações em cada classe) pelo número total de observações no conjunto de dados. Essa coluna fornece a proporção de observações em cada classe em relação ao total.

3. `nova_dist_freq['Frequência acumulada'] = nova_dist_freq['Frequência'].cumsum()`: Aqui, você está calculando a frequência acumulada. A frequência acumulada é a soma cumulativa das frequências absolutas à medida que você percorre as classes. Isso significa que cada linha na coluna "Frequência acumulada" representa a soma das frequências de todas as classes anteriores, incluindo a frequência da classe atual.

Essas duas colunas adicionadas são úteis para análises estatísticas e construção de gráficos, pois ajudam a entender a distribuição dos dados em relação às classes. A frequência relativa mostra a proporção de dados em cada classe, enquanto a frequência acumulada ajuda a visualizar como as frequências se acumulam à medida que você avança nas classes.

Construção do histograma de frequências:

```
import matplotlib.pyplot as plt

# Construção do histograma
histograma = data.hist(bins=[inter.left for inter in intervalos]+[v_max],
                       color='blue', edgecolor='black', grid=False)
# Inserção de atributos ao gráfico
histograma.set(xlabel='Preço [US$]',
               ylabel='Frequência - Número de navegadores GPS',
               title='Distribuição de frequências dos preços de GPS',
               xticks=nova_dist_freq['Pontos médios'],
               yticks=range(0, nova_dist_freq['Frequência'].max()+2, 2))
plt.show()
```

Neste trecho de código, você está construindo um histograma para visualizar a distribuição de

frequências dos preços de GPS. Aqui está a explicação de cada linha:

1. `histograma = data.hist(bins=[inter.left for inter in intervalos]+[v_max], color='blue', edgecolor='black', grid=False)`: Você está usando o método `hist` do Pandas para criar o histograma. Aqui estão os argumentos e o que eles fazem:
   - `bins`: Este argumento especifica os intervalos das classes que você deseja usar no histograma. Os intervalos foram definidos com base nos seus cálculos anteriores e na variável "intervalos".
   - `color`: Define a cor das barras do histograma.
   - `edgecolor`: Define a cor das bordas das barras do histograma.
   - `grid`: Define se as linhas de grade devem ser exibidas no gráfico. Neste caso, está definido como "False" para desativar as linhas de grade.

2. `histograma.set(...)`: Este trecho é usado para definir atributos do gráfico, como rótulos dos eixos, título e marcações nos eixos. Aqui estão os detalhes:
   - `xlabel`: Define o rótulo do eixo x como "Preço [US$]".
   - `ylabel`: Define o rótulo do eixo y como "Frequência - Número de navegadores GPS".
   - `title`: Define o título do gráfico como "Distribuição de frequências dos preços de GPS".
   - `xticks`: Define as marcações no eixo x usando os pontos médios das classes calculados anteriormente.
   - `yticks`: Define as marcações no eixo y de 0 ao valor máximo da frequência mais 2, em incrementos de 2.

3. `plt.show()`: Este comando exibe o gráfico do histograma na tela.

No geral, este trecho de código cria um histograma informativo para visualizar como os preços dos navegadores GPS estão distribuídos em diferentes intervalos de classe. Isso ajuda a entender a distribuição dos dados em relação aos preços.

```
# Construção do histograma
histograma = data.hist(bins=[inter.left for inter in intervalos]+[v_max],
                       color='green', edgecolor='black', grid=False)
# Inserção de atributos ao gráfico
histograma.set(xlabel='Preço [US$]',
               ylabel='Frequência - Número de navegadores GPS',
               title='Distribuição de frequências dos preços de GPS',
               xticks=nova_dist_freq['Pontos médios'],
               yticks=range(0,nova_dist_freq['Frequência'].max()+2,2))
# Inserção de rótulos nas barras do histograma
barras = histograma.patches
freqs = nova_dist_freq['Frequência']
for barra, freq in zip(barras, freqs):
    altura = barra.get_height()

plt.show()
```

Neste trecho de código, você está construindo um histograma semelhante ao anterior, mas agora adicionando rótulos nas barras do histograma para mostrar a frequência exata em cada classe. Vou explicar cada linha:

1. `histograma = data.hist(bins=[inter.left for inter in intervalos]+[v_max], color='green', edgecolor='black', grid=False)`: Você está criando o histograma com os mesmos argumentos que no exemplo anterior. O único detalhe novo aqui é que você está definindo a cor das barras como verde.

2. `histograma.set(...)`: Assim como no exemplo anterior, você está definindo rótulos para os eixos x e y, título e marcações nos eixos.

3. `barras = histograma.patches`: Aqui, você está obtendo uma lista de todas as barras (retângulos) no histograma. Cada barra representa uma classe de intervalo.

4. `freqs = nova_dist_freq['Frequência']`: Você está obtendo a coluna de frequência do DataFrame "nova_dist_freq", que contém as frequências de cada classe.

5. Em seguida, você entra em um loop para percorrer cada barra e sua frequência correspondente:
   - `for barra, freq in zip(barras, freqs):`: O comando `zip` permite que você percorra as listas `barras` e `freqs` simultaneamente.

6. Dentro do loop, você está calculando a altura da barra com `altura = barra.get_height()`. Isso obtém a altura da barra do histograma.

O objetivo principal deste trecho de código é adicionar as frequências como rótulos nas barras do histograma, para que você possa visualizar diretamente o número exato de observações em cada classe. Essa é uma maneira útil de enriquecer a visualização do histograma com informações quantitativas.

Construção do polígono de frequências:

```
# Cálculo dos pontos médios fictícios
fic_esq = [pts_medios[0]-amplitude]
print(f'Ponto médio fictício à esquerda = {fic_esq[0]}')
fic_dir = [pts_medios[-1]+amplitude]
print(f'Ponto médio fictício à direita = {fic_dir[0]}')
```

Neste trecho de código, você está calculando os pontos médios fictícios à esquerda e à direita das classes para a construção do histograma. Aqui estão as explicações para cada parte do código:

1. `fic_esq = [pts_medios[0]-amplitude]`: Você está calculando o ponto médio fictício à esquerda do primeiro intervalo de classe. Isso é feito subtraindo a amplitude (que você calculou anteriormente) do ponto médio da primeira classe. O resultado é armazenado na lista `fic_esq`.

2. `print(f'Ponto médio fictício à esquerda = {fic_esq[0]}')`: Você está imprimindo o valor do ponto médio fictício à esquerda. O f-string (`f'...'`) é usado para formatar a saída, incluindo o valor calculado.

3. `fic_dir = [pts_medios[-1]+amplitude]`: Você está calculando o ponto médio fictício à direita do último intervalo de classe. Isso é feito adicionando a amplitude ao ponto médio da última classe. O resultado é armazenado na lista `fic_dir`.

4. `print(f'Ponto médio fictício à direita = {fic_dir[0]}')`: Você está imprimindo o valor do ponto médio fictício à direita. Mais uma vez, o f-string é usado para formatar a saída.

Esses pontos médios fictícios são usados na construção do histograma para garantir que a primeira e a última classe tenham barras que vão até o valor mínimo e máximo dos dados, respectivamente. Eles ajudam a criar uma representação visual precisa da distribuição dos dados, mesmo quando não há observações exatamente no limite das classes.

```
# Construção dos dados com inserção dos pontos médios fictícios
x_data = fic_esq + list(nova_dist_freq['Pontos médios']) + fic_dir
print(f'Valores para eixo x: {x_data}')
y_data = [0] + list(nova_dist_freq['Frequência']) + [0]
print(f'Valores para eixo y: {y_data}')
```

Neste trecho de código, você está construindo os dados que serão usados para criar um histograma com pontos médios fictícios. Aqui estão as explicações para cada parte do código:

1. `x_data = fic_esq + list(nova_dist_freq['Pontos médios']) + fic_dir`: Você está criando uma lista `x_data` que contém os valores que serão plotados no eixo x do histograma. Essa lista começa com o ponto médio fictício à esquerda (calculado anteriormente) e é seguida pelos pontos médios reais das classes (extraídos do DataFrame `nova_dist_freq`) e, por fim, pelo ponto médio fictício à direita (também calculado anteriormente). Isso garante que o histograma inclua barras para as classes fictícias nas extremidades.

2. `print(f'Valores para eixo x: {x_data}')`: Você está imprimindo os valores que serão usados no eixo x do histograma. O f-string (`f'...'`) é usado para formatar a saída, incluindo os valores contidos na lista `x_data`.

3. `y_data = [0] + list(nova_dist_freq['Frequência']) + [0]`: Você está criando uma lista `y_data` que contém os valores que serão plotados no eixo y do histograma. Essa lista começa com zero (0), uma vez que não há frequência associada aos pontos médios fictícios à esquerda e à direita. Em seguida, você inclui as frequências reais das classes, que são extraídas do DataFrame `nova_dist_freq`, e finalmente, termina com zero (0) novamente para a classe fictícia à direita.

Esses dados são essenciais para construir um histograma adequado que represente a distribuição de frequências dos dados, incluindo as classes fictícias nas extremidades para garantir uma representação visual precisa.

```
# Plotagem do polígono de frequências, com diversos atributos
plt.plot(x_data, y_data,'go--')
plt.xlabel('Preço [US$]')
plt.ylabel('Frequência - Número de navegadores GPS')
plt.title('Distribuição de frequências dos preços de GPS')
plt.xticks(x_data)
plt.yticks(range(0, nova_dist_freq['Frequência'].max()+2, 2))
plt.grid()
```

Neste trecho de código, você está plotando o polígono de frequências usando a função `plt.plot()` do Matplotlib. Aqui estão as explicações para cada parte do código:

1. `plt.plot(x_data, y_data,'go--')`: Esta linha de código cria o gráfico de linhas com marcadores.
   - `x_data` e `y_data` são os dados que você definiu anteriormente para o eixo x e y.
   - `'go--'` é um argumento de formatação que diz ao Matplotlib como desenhar o gráfico. Aqui está o que cada parte significa:
     - `'g'`: Define a cor da linha para verde (green).
     - `'o'`: Define os marcadores como círculos.
     - `'--'`: Define o estilo da linha como tracejada.

2. `plt.xlabel('Preço [US$]')`: Define um rótulo para o eixo x do gráfico, indicando que ele representa os preços em dólares americanos.

3. `plt.ylabel('Frequência - Número de navegadores GPS')`: Define um rótulo para o eixo y do gráfico, indicando que ele representa a frequência, ou seja, o número de navegadores GPS.

4. `plt.title('Distribuição de frequências dos preços de GPS')`: Define um título para o gráfico, descrevendo o que ele representa.

5. `plt.xticks(x_data)`: Define os valores do eixo x com base nos dados contidos em `x_data`. Isso garante que os valores no eixo x correspondam aos pontos médios das classes e às classes fictícias nas extremidades.

6. `plt.yticks(range(0, nova_dist_freq['Frequência'].max()+2, 2))`: Define os valores do eixo y com base nos dados contidos em `nova_dist_freq`. Isso ajuda a ajustar automaticamente os rótulos no eixo y de acordo com a frequência máxima da distribuição.

7. `plt.grid()`: Adiciona uma grade ao gráfico para facilitar a leitura das coordenadas.

Em resumo, esse trecho de código configura e plota o gráfico de frequências usando as informações previamente preparadas, tornando-o informativo e fácil de entender.

```
# Plotagem do polígono de frequências com frequências relativas
y_data_rel = [y/sum(y_data) for y in y_data]
plt.plot(x_data, y_data_rel,'ro--')
plt.xlabel('Preço [US$]')
plt.ylabel('Frequência relativa')
plt.title('Distribuição de frequências dos preços de GPS')
plt.xticks(x_data)
plt.grid()
```

Neste trecho de código, você está plotando o polígono de frequências relativas. Aqui estão as explicações para cada parte do código:

1. `y_data_rel = [y/sum(y_data) for y in y_data]`: Esta linha de código cria uma nova lista chamada `y_data_rel`, na qual cada valor de frequência em `y_data` é dividido pela soma total de todos os valores em `y_data`. Isso calcula as frequências relativas, que representam a proporção de cada classe em relação ao total.

2. `plt.plot(x_data, y_data_rel,'ro--')`: Esta linha de código cria o gráfico de linhas com marcadores para as frequências relativas.
   - `x_data` e `y_data_rel` são os dados usados para o eixo x e y, respectivamente.
   - `'ro--'` é um argumento de formatação que diz ao Matplotlib como desenhar o gráfico. Aqui está o que cada parte significa:
     - `'r'`: Define a cor da linha para vermelho (red).
     - `'o'`: Define os marcadores como círculos.
     - `'--'`: Define o estilo da linha como tracejada.

3. `plt.xlabel('Preço [US$]')`: Define um rótulo para o eixo x do gráfico, indicando que ele representa os preços em dólares americanos.

4. `plt.ylabel('Frequência relativa')`: Define um rótulo para o eixo y do gráfico, indicando que ele representa as frequências relativas.

5. `plt.title('Distribuição de frequências dos preços de GPS')`: Define um título para o gráfico, descrevendo o que ele representa.

6. `plt.xticks(x_data)`: Define os valores do eixo x com base nos dados contidos em `x_data`. Isso garante que os valores no eixo x correspondam aos pontos médios das classes e às classes fictícias nas extremidades.

7. `plt.grid()`: Adiciona uma grade ao gráfico para facilitar a leitura das coordenadas.

Em resumo, esse trecho de código cria um gráfico de frequências relativas, que representa a proporção de cada classe em relação ao total. Isso ajuda a visualizar a distribuição de frequências de forma normalizada, onde a área total sob a curva é igual a 1.