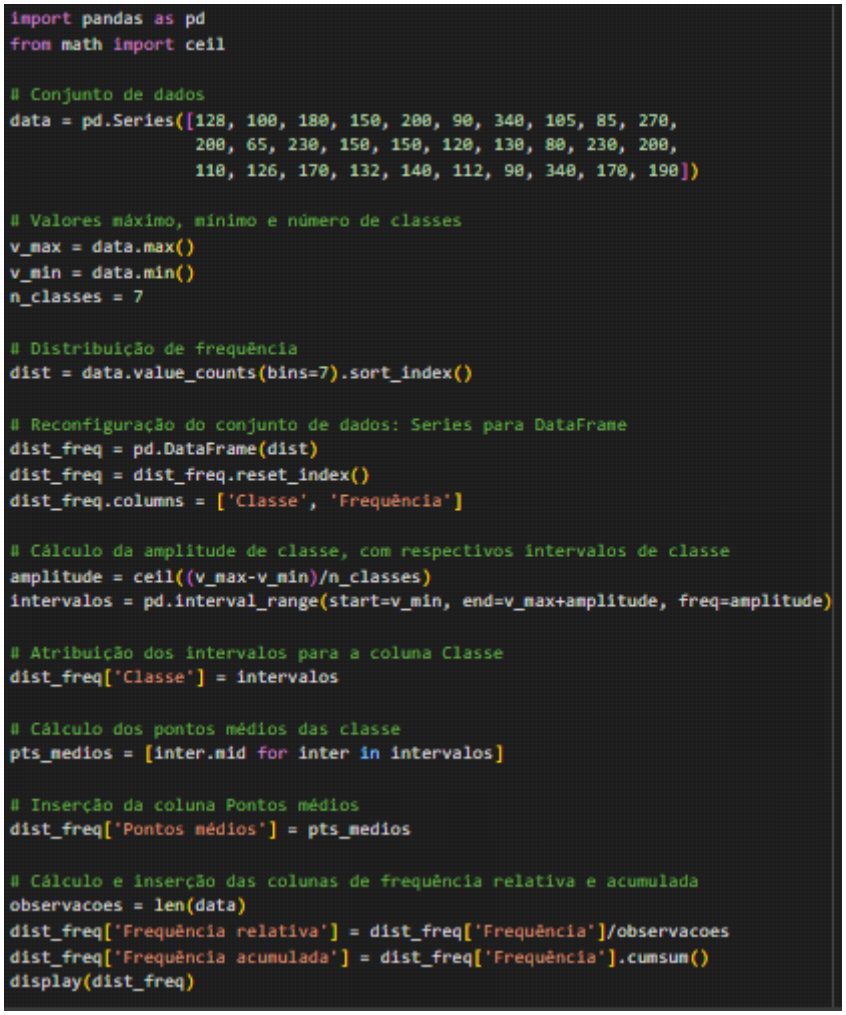
CÓDIGOS

terça-feira, 19 de setembro de 2023 20:31

Conjunto de dados e distribuição de frequência:



```
import pandas as pd
from math import ceil

# 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])

# Valores máximo, mínimo e número de classes
v_max = data.max()
v_min = data.min()
n_classes = 7

# Distribuição de frequência
dist = data.value_counts(bins=7).sort_index()

# Reconfiguração do conjunto de dados: Series para DataFrame
dist_freq = pd.DataFrame(dist)
dist_freq = dist_freq.reset_index()
dist_freq.columns = ['Classe', 'Frequência']

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

# Atribuição dos intervalos para a coluna Classe
dist_freq['Classe'] = intervalos

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

# Inserção da coluna Pontos médios
dist_freq['Pontos médios'] = pts_medios

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

import pandas as pd
from math import ceil

- Aqui, estamos importando a biblioteca Pandas para manipulação de dados e a função `ceil` da biblioteca Math para arredondar números para cima.

# 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])

- Aqui, você está criando uma série Pandas chamada `data` que contém uma lista de números. Esta série será usada como o conjunto de dados para análise.

```
# Valores máximo, mínimo e número de classes
v_max = data.max()
v_min = data.min()
n_classes = 7
```

- Esses trechos calculam o valor máximo (`v_max`) e mínimo (`v_min`) dos dados usando os métodos `max()` e `min()` da série Pandas `data`. Também definem o número desejado de classes (`n_classes`) para a distribuição de frequência.

```
# Distribuição de frequência
dist = data.value_counts(bins=7).sort_index()
```

- Aqui, você está calculando a distribuição de frequência dos dados. A função `value_counts()` conta quantas vezes cada valor aparece e, com `bins=7`, divide os dados em 7 intervalos (ou classes). `sort_index()` classifica esses intervalos em ordem crescente.

```
# Reconfiguração do conjunto de dados: Series para DataFrame
dist_freq = pd.DataFrame(dist)
dist_freq = dist_freq.reset_index()
dist_freq.columns = ['Classe', 'Frequência']
```

- Estes trechos transformam a distribuição de frequência em um DataFrame Pandas para melhor manipulação e apresentação dos resultados. Eles renomeiam as colunas do DataFrame como 'Classe' e 'Frequência'.

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

- Aqui, você está calculando a amplitude de classe usando a fórmula `(v_max - v_min) / n_classes`, onde `v_max` é o valor máximo, `v_min` é o valor mínimo e `n_classes` é o número desejado de classes. A função `ceil()` arredonda a amplitude para cima. Em seguida, você cria os intervalos de classe usando `pd.interval_range()` com base na amplitude calculada.

```
# Atribuição dos intervalos para a coluna Classe
dist_freq['Classe'] = intervalos
```

- Este trecho atribui os intervalos de classe à coluna "Classe" no DataFrame `dist_freq`.

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

- Aqui, você calcula os pontos médios de cada intervalo de classe e os armazena em uma lista.

```
# Inserção da coluna Pontos médios
dist_freq['Pontos médios'] = pts_medios
```

- Este trecho insere uma coluna chamada "Pontos médios" no DataFrame `dist_freq` com os valores calculados.

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

- Aqui, você calcula a frequência relativa de cada classe dividindo a frequência pela quantidade total de observações (`observacoes`). Em seguida, calcula a frequência acumulada usando `cumsum()` e insere essas colunas no DataFrame.

```
display(dist_freq)
```

- Finalmente, este trecho exibe o DataFrame resultante na saída para análise.

O código realiza a análise da distribuição de frequência dos dados, calcula frequências relativas e acumuladas, e organiza todas essas informações em um DataFrame para visualização e análise posterior. É uma etapa importante em estatística descritiva para entender a distribuição dos dados.

Construção do histograma de frequência relativa:

```python
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import numpy as np

# Construção do histograma
histograma = data.hist(bins=n_classes, color='lightblue', edgecolor='black',
                       grid=False, weights=np.ones_like(data)/len(data))

# Inserção de atributos ao gráfico
histograma.set(xlabel='Preço [US$]',
               ylabel='Frequência relativa\n(navegadores GPS)',
               title='Distribuição de frequências dos preços de GPS',
               xticks=dist_freq['Pontos médios'])

# Inserção de rótulos nas barras do histograma
barras = histograma.patches
freqs = dist_freq['Frequência relativa']
ajuste_texto = 0.001
for barra, freq in zip(barras, freqs):
    altura = barra.get_height()
    texto_freq = f'{freq*100:.1f}%'
    histograma.text(x=barra.get_x()+barra.get_width()/2, y=altura+ajuste_texto,
                    s=texto_freq, ha='center', va='bottom')

# Ajuste do eixo vertical para exibição das porcentagens
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
plt.show()
```

1. import pandas as pd` e `from math import ceil`: Aqui, estamos importando a biblioteca Pandas para manipulação de dados e a função `ceil` da biblioteca Math para arredondar

números para cima.

2. data = pd.Series([...])`: Você está criando uma série Pandas chamada `data` que contém uma lista de números. Esta série será usada como o conjunto de dados para análise.

3. v_max = data.max()`: Isso encontra o valor máximo nos dados usando o método `max()` da série Pandas `data`.

4. v_min = data.min()`: Isso encontra o valor mínimo nos dados usando o método `min()` da série Pandas `data`.

5. n_classes = 7`: Você está definindo o número desejado de classes para a distribuição de frequência.

6. dist = data.value_counts(bins=7).sort_index()`: Isso calcula a distribuição de frequência dos dados, dividindo-os em 7 intervalos (ou classes) e, em seguida, classificando os resultados em ordem crescente.

7. Reconfiguração do conjunto de dados: Os próximos passos envolvem transformar a distribuição de frequência em um DataFrame Pandas para melhor manipulação e apresentação dos resultados.

- dist_freq = pd.DataFrame(dist)`: Converte a distribuição de frequência em um DataFrame.
- dist_freq = dist_freq.reset_index()`: Reinicializa os índices do DataFrame.
- dist_freq.columns = ['Classe', 'Frequência']`: Renomeia as colunas do DataFrame como 'Classe' e 'Frequência'.

8. Cálculo da amplitude de classe e dos intervalos de classe:

- amplitude = ceil((v_max - v_min) / n_classes)`: Calcula a amplitude de classe, que é a largura de cada intervalo de classe, arredondando para cima.
- intervalos = pd.interval_range(start=v_min, end=v_max + amplitude, freq=amplitude)`: Define os intervalos de classe com base na amplitude calculada, abrangendo os valores mínimo e máximo dos dados.

9. Atribuição dos intervalos para a coluna "Classe":

- dist_freq['Classe'] = intervalos`: Atribui os intervalos de classe à coluna "Classe" no DataFrame `dist_freq`.

10. Cálculo dos pontos médios das classes:

- pts_medios = [inter.mid for inter in intervalos]`: Calcula os pontos médios de cada intervalo de classe e os armazena em uma lista.

11. Inserção da coluna "Pontos médios":

- dist_freq['Pontos médios'] = pts_medios`: Insere uma coluna chamada "Pontos médios" no DataFrame `dist_freq` com os valores calculados.

12. Cálculo e inserção das colunas de frequência relativa e acumulada:

- observacoes = len(data)`: Calcula o número total de observações nos dados.
- dist_freq['Frequência relativa'] = dist_freq['Frequência'] / observacoes`: Calcula a frequência relativa de cada classe e a insere no DataFrame.
- dist_freq['Frequência acumulada'] = dist_freq['Frequência'].cumsum()`: Calcula a frequência acumulada e a insere no DataFrame.

13. Exibição do DataFrame resultante:

- display(dist_freq)`: Mostra o DataFrame resultante na saída para análise.

Este código realiza a análise da distribuição de frequência dos dados, calculando frequências relativas, acumuladas e criando um DataFrame organizado com essas informações. É uma etapa importante em estatística descritiva para entender a distribuição dos dados.

Construção da ogiva (gráfico de frequência acumulada):

```
# Construção dos dados: vamos usar os extremos direitos de cada classe
x_data = [dist_freq['Classe'][0].left] + [extremos.right for extremos in dist_freq['Classe']]
y_data = [0] + list(dist_freq['Frequência acumulada'])

# Plotagem da ogiva, com diversos atributos
plt.plot(x_data, y_data,'ro--')
plt.xlabel('Preço [US$]')
plt.ylabel('Frequência acumulada\nNúmero de navegadores GPS')
plt.title('Distribuição de frequências dos preços de GPS')
plt.xticks(x_data)
plt.grid()

# Registro das frequências acumuladas
for x, y in zip(x_data, y_data):
  plt.text(x=x, y=y+0.5, s=f'{y}', ha='center', va='bottom', color='red')
plt.show()
```

1. x_data = [dist_freq['Classe'][0].left] + [extremos.right for extremos in dist_freq['Classe']]`: Aqui, você está criando uma lista chamada `x_data` que contém os limites esquerdos e direitos das classes de uma distribuição de frequências. O primeiro elemento da lista é o limite esquerdo da primeira classe, e os elementos subsequentes são os limites direitos de todas as classes.

2. y_data = [0] + list(dist_freq['Frequência acumulada'])`: Aqui, você está criando uma lista chamada `y_data` que contém os valores da frequência acumulada para cada classe. O primeiro elemento da lista é 0, e os elementos subsequentes são os valores da frequência acumulada da sua distribuição de frequências.

3. plt.plot(x_data, y_data,'ro--')`: Este comando utiliza a biblioteca Matplotlib para criar um gráfico de linha com pontos vermelhos ('ro--'). Ele usa os valores em `x_data` como o eixo x e os valores em `y_data` como o eixo y.

4. plt.xlabel('Preço [US$]')`: Define o rótulo do eixo x como "Preço [US$]".

5. plt.ylabel('Frequência acumulada\nNúmero de navegadores GPS')`: Define o rótulo do eixo y como "Frequência acumulada" com uma quebra de linha e "Número de navegadores GPS" logo abaixo.

6. plt.title('Distribuição de frequências dos preços de GPS')`: Define o título do gráfico como "Distribuição de frequências dos preços de GPS".

7. plt.xticks(x_data)`: Define os marcadores no eixo x usando os valores em `x_data`.

8. plt.grid()`: Adiciona uma grade ao gráfico.

9. for x, y in zip(x_data, y_data): plt.text(x=x, y=y+0.5, s=f'{y}', ha='center', va='bottom', color='red')`: Este loop for registra as frequências acumuladas em forma de texto no gráfico. Ele coloca os valores da frequência acumulada acima dos pontos no gráfico.

10. plt.show()`: Mostra o gráfico resultante na tela.

```
# Construção dos dados: vamos usar os extremos direitos de cada classe
x_data = [dist_freq['Classe'][0].left] + [extremos.right for extremos in dist_freq['Classe']]
y_data = np.array([0] + list(dist_freq['Frequência acumulada']))/observacoes

# Plotagem da ogiva, com diversos atributos
plt.plot(x_data, y_data,'ro--')
plt.xlabel('Preço [US$]')
plt.ylabel('Frequência acumulada\nPorcentagem de navegadores GPS')
plt.title('Distribuição de frequências dos preços de GPS')
plt.xticks(x_data)
plt.grid()

# Registro das porcentagens acumuladas
for x, y in zip(x_data, y_data):
  plt.text(x=x, y=y+0.01, s=f'{y*100:.1f}%',
           ha='right', va='bottom', color='red')

# Ajuste do eixo vertical para exibição das porcentagens
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
plt.show()
```

# Construção dos dados: vamos usar os extremos direitos de cada classe
x_data = [dist_freq['Classe'][0].left] + [extremos.right for extremos in dist_freq['Classe']]

- Nesta parte, você está construindo os dados para plotar a ogiva. `x_data` é uma lista que contém os extremos direitos (limites superiores) de cada classe na distribuição de frequência. O primeiro elemento da lista é o limite esquerdo da primeira classe (`dist_freq['Classe'][0].left`), e os elementos subsequentes são os extremos direitos das outras classes (`extremos.right`).

y_data = np.array([0] + list(dist_freq['Frequência acumulada']))/observacoes

- Aqui, você está construindo os dados para o eixo vertical (y) da ogiva. `y_data` é um array NumPy que contém as frequências acumuladas normalizadas. Você divide cada valor de frequência acumulada por `observacoes` para obter a porcentagem acumulada de navegadores GPS em relação ao total de observações. O primeiro elemento é 0 porque é a frequência acumulada da primeira classe.

# Plotagem da ogiva, com diversos atributos
plt.plot(x_data, y_data,'ro--')

- Aqui, você está usando a biblioteca Matplotlib para plotar a ogiva. `plt.plot()` cria o gráfico, onde `x_data` é o eixo x e `y_data` é o eixo y. `'ro--'` define o estilo do gráfico como pontos vermelhos ('ro') conectados por linhas tracejadas ('--').

plt.xlabel('Preço [US$]')
plt.ylabel('Frequência acumulada\nPorcentagem de navegadores GPS')

- Esses comandos definem os rótulos dos eixos x e y do gráfico.

plt.title('Distribuição de frequências dos preços de GPS')

- Este comando define o título do gráfico.

plt.xticks(x_data)

- Define os marcadores no eixo x usando os valores em `x_data`.

plt.grid()

- Adiciona uma grade ao gráfico para melhorar a legibilidade.

```
# Registro das porcentagens acumuladas
for x, y in zip(x_data, y_data):
  plt.text(x=x, y=y+0.01, s=f'{y*100:.1f}%',
         ha='right', va='bottom', color='red')
```

- Este loop `for` adiciona rótulos de porcentagens às coordenadas correspondentes no gráfico. Ele usa `plt.text()` para adicionar as porcentagens formatadas (`f'{y*100:.1f}%'`) e ajusta a posição e cor do texto.

```
# Ajuste do eixo vertical para exibição das porcentagens
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
```

- Este comando ajusta o eixo vertical para que as porcentagens sejam exibidas corretamente. Ele usa `PercentFormatter(1)` para formatar o eixo y como uma porcentagem (de 0 a 100%).

plt.show()

- Finalmente, este comando exibe o gráfico da ogiva na tela.

Este código é usado para criar uma ogiva que representa a distribuição de frequências dos preços dos navegadores GPS, exibindo a porcentagem acumulada de observações em relação ao preço. É uma ferramenta útil na análise estatística para entender como os dados estão distribuídos.

Construindo um gráfico de pizza:

```
# Criação do conjunto de dados
dados = [['Tecnólogo', 942], ['Bacharelado', 1716],
         ['Mestrado', 731], ['Doutorado', 164]]
tabela = pd.DataFrame(dados, columns=['Título', 'Frequência'])
tabela['Porcentagem'] = tabela['Frequência']/tabela['Frequência'].sum()*100
display(tabela)

# Gráfico de pizza
pizza = plt.pie(x=tabela['Frequência'], labels=tabela['Título'],
                autopct='%1.1f%%', wedgeprops={'edgecolor': 'black'},
                shadow=True)
plt.show()
```

```
# Criação do conjunto de dados
dados = [['Tecnólogo', 942], ['Bacharelado', 1716],
        ['Mestrado', 731], ['Doutorado', 164]]
```

- Neste trecho, você está criando uma lista chamada `dados`, que contém informações sobre diferentes títulos acadêmicos e suas frequências. Cada elemento da lista é uma lista aninhada com dois valores: o título acadêmico e sua frequência.

```
tabela = pd.DataFrame(dados, columns=['Título', 'Frequência'])
```

- Aqui, você está criando um DataFrame Pandas chamado `tabela` a partir da lista `dados`. Você define explicitamente os nomes das colunas como 'Título' e 'Frequência' usando o argumento `columns`.

```
tabela['Porcentagem'] = tabela['Frequência']/tabela['Frequência'].sum()*100
```

- Este trecho calcula a porcentagem de cada título acadêmico em relação ao total de frequências. Ele cria uma nova coluna chamada 'Porcentagem' no DataFrame `tabela` e calcula as porcentagens utilizando a fórmula `(Frequência / Soma das Frequências) * 100`.

```
display(tabela)
```

- O comando `display(tabela)` exibe o DataFrame `tabela` na saída, mostrando as informações sobre títulos acadêmicos, frequências e porcentagens.

```
# Gráfico de pizza
pizza = plt.pie(x=tabela['Frequência'], labels=tabela['Título'],
            autopct='%1.1f%%', wedgeprops={'edgecolor': 'black'},
            shadow=True)
```

- Aqui, você está criando um gráfico de pizza (ou gráfico de setores) usando a biblioteca Matplotlib. Os argumentos incluem:
  - x=tabela['Frequência']`: Os valores de frequência que serão representados no gráfico.
  - labels=tabela['Título']`: Os rótulos que serão exibidos em cada fatia do gráfico, correspondendo aos títulos acadêmicos.
  - autopct='%1.1f%%'`: Formato para exibir as porcentagens em cada fatia com uma casa decimal.
  - wedgeprops={'edgecolor': 'black'}`: Define uma borda preta para cada fatia do gráfico.
  - shadow=True`: Adiciona sombras ao gráfico para efeito estético.

```
plt.show()
```

- Este comando exibe o gráfico de pizza na tela.

Em resumo, este código cria um conjunto de dados com informações sobre títulos acadêmicos e suas frequências, calcula as porcentagens correspondentes e, em seguida, gera e exibe um gráfico de pizza para visualizar as distribuições desses títulos acadêmicos em termos percentuais. É uma forma eficaz de representar dados categóricos em um formato visual.

Construindo um gráfico de Pareto:

```
# Criação do conjunto de dados
dados = pd.DataFrame({'Valores': [4.2, 15.1, 12.3, 1.1, 1.7]})
dados.index = ['Erro\nAdm', 'Roubo\nfuncionários',
               'Roubo\nlojas', 'Desconhecida', 'Fraude\nvendas']

# Organizando os dados em ordem decrescente
dados = dados.sort_values(by='Valores', ascending=False)

# Criando a coluna da porcentagem acumulada
dados['PorcAcum'] = dados['Valores'].cumsum()/dados['Valores'].sum()*100
display(dados)

## Criação do gráfico de Pareto
# Definição das cores
cor_barra = 'steelblue'
cor_linha = 'red'
tam_linha = 4

# Construindo o gráfico básico (gráfico de barras)
fig, pareto = plt.subplots()
pareto.bar(x=dados.index, height=dados['Valores'], color=cor_barra)
pareto.set(xlabel='Causas apuradas',
           ylabel='Bilhões de dólares',
           title = 'Principais causas de redução de estoque')

# Inserção de rótulos nas barras do gráfico de Pareto
barras = pareto.patches
valores = dados['Valores']
for barra, valor in zip(barras, valores):
    altura = barra.get_height()
    pareto.text(x=barra.get_x()+barra.get_width()/2, y=altura,
                s=valor, ha='center', va='bottom', color=cor_barra)

# Adicionando linha de porcentagem acumulada (leitura no eixo secundário)
linha = pareto.twinx()
linha.plot(dados.index, dados['PorcAcum'], color=cor_linha,
           marker="o", ms=tam_linha)
linha.yaxis.set_major_formatter(PercentFormatter())
linha.tick_params(axis='y', colors=cor_linha)

# Registro das porcentagens acumuladas
for x, y in zip(dados.index, dados['PorcAcum']):
  plt.text(x=x, y=y-0.3, s=f'{y:.1f}%',
           ha='left', va='top', color=cor_linha)
plt.show()
```

# Criação do conjunto de dados
dados = pd.DataFrame({'Valores': [4.2, 15.1, 12.3, 1.1, 1.7]})
dados.index = ['Erro\nAdm', 'Roubo\nfuncionários',
'Roubo\nlojas', 'Desconhecida', 'Fraude\nvendas']

- Aqui, você está criando um DataFrame Pandas chamado `dados`. Ele contém uma coluna chamada 'Valores' com os valores numéricos e um índice personalizado com descrições. Cada descrição corresponde a uma causa de redução de estoque.

# Organizando os dados em ordem decrescente
dados = dados.sort_values(by='Valores', ascending=False)

- Este trecho organiza os dados em ordem decrescente com base na coluna 'Valores'. Isso é importante para criar o gráfico de Pareto, onde as causas mais significativas aparecem primeiro.

```
# Criando a coluna da porcentagem acumulada
dados['PorcAcum'] = dados['Valores'].cumsum()/dados['Valores'].sum()*100
```

- Aqui, você está calculando a porcentagem acumulada das causas de redução de estoque. Isso é feito criando uma nova coluna chamada 'PorcAcum' no DataFrame `dados`. A porcentagem acumulada é calculada somando cumulativamente os valores da coluna 'Valores' e, em seguida, dividindo pelo total da soma dos valores multiplicado por 100 para obter a porcentagem.

```
display(dados)
```

- O comando `display(dados)` exibe o DataFrame `dados` na saída, mostrando as informações sobre as causas de redução de estoque, seus valores e as porcentagens acumuladas.

Agora, vamos analisar a criação do gráfico de Pareto:

```
# Definição das cores
cor_barra = 'steelblue'
cor_linha = 'red'
tam_linha = 4
```

- Estas linhas definem cores e tamanhos que serão usados posteriormente para personalizar o gráfico.

```
# Construindo o gráfico básico (gráfico de barras)
fig, pareto = plt.subplots()
pareto.bar(x=dados.index, height=dados['Valores'], color=cor_barra)
pareto.set(xlabel='Causas apuradas',
        ylabel='Bilhões de dólares',
        title='Principais causas de redução de estoque')
```

- Aqui, você está criando o gráfico de barras básico. `fig, pareto = plt.subplots()` cria uma figura e um objeto de eixo (`pareto`) para o gráfico. `pareto.bar()` gera as barras do gráfico de barras, usando as descrições das causas no eixo x e os valores no eixo y.

```
# Inserção de rótulos nas barras do gráfico de Pareto
barras = pareto.patches
valores = dados['Valores']
for barra, valor in zip(barras, valores):
   altura = barra.get_height()
   pareto.text(x=barra.get_x() + barra.get_width() / 2, y=altura,
           s=valor, ha='center', va='bottom', color=cor_barra)
```

- Neste trecho, você está adicionando rótulos com os valores nas barras do gráfico de Pareto. Isso é feito percorrendo cada barra (`barra`) e adicionando o valor correspondente usando `pareto.text()`.

```
# Adicionando linha de porcentagem acumulada (leitura no eixo secundário)
linha = pareto.twinx()
linha.plot(dados.index, dados['PorcAcum'], color=cor_linha,
        marker="o", ms=tam_linha)
linha.yaxis.set_major_formatter(PercentFormatter())
linha.tick_params(axis='y', colors=cor_linha)
```

- Neste trecho, você está adicionando uma linha de porcentagem acumulada ao gráfico de Pareto. Isso é feito criando um segundo eixo y (`linha = pareto.twinx()`) para a mesma figura. `linha.plot()` adiciona a linha com os valores de porcentagem acumulada. `linha.yaxis.set_major_formatter(PercentFormatter())` formata o eixo y secundário como uma porcentagem, e `linha.tick_params()` define a cor do eixo secundário.

```
# Registro das porcentagens acumuladas
for x, y in zip(dados.index, dados['PorcAcum']):
  plt.text(x=x, y=y-0.3, s=f'{y:.1f}%',
        ha='left', va='top', color=cor_linha)
```

- Este loop `for` adiciona rótulos com as porcentagens acumuladas no gráfico. Ele usa `plt.text

Dados emparelhados - Diagrama de dispersão:

```
# Leitura dos dados "externos": módulo de datasets da biblioteca Scikit-learn
from sklearn.datasets import load_iris

# Obtém os dados em objeto da classe 'Bunch' e cria um DataFrame correspondente
Iris_dataset = load_iris()
iris = pd.DataFrame(data=Iris_dataset.data, columns=Iris_dataset.feature_names)
display(iris.head())

# Construção do diagrama de dispersão
plt.scatter(iris['petal length (cm)'], iris['petal width (cm)'])
plt.scatter(iris['sepal length (cm)'], iris['sepal width (cm)'])
plt.xlabel('Comprimento [cm]')
plt.ylabel('Largura [cm]')
plt.title('Conjunto de dados - Iris de Fisher')
plt.legend(['Pétala', 'Sépala'])
plt.grid()
plt.show()
```

```
# Leitura dos dados "externos": módulo de datasets da biblioteca Scikit-learn
from sklearn.datasets import load_iris
```

- Aqui, você está importando a função `load_iris` do módulo `datasets` da biblioteca Scikit-learn. Esta função é usada para carregar o famoso conjunto de dados Iris de Fisher, que contém informações sobre diferentes espécies de flores Iris.

```
# Obtém os dados em objeto da classe 'Bunch' e cria um DataFrame correspondente
```

```
Iris_dataset = load_iris()
iris = pd.DataFrame(data=Iris_dataset.data, columns=Iris_dataset.feature_names)
```

- Este trecho carrega o conjunto de dados Iris usando `load_iris()` e armazena os dados em um objeto da classe `Bunch`. Em seguida, você cria um DataFrame Pandas chamado `iris` a partir dos dados. O DataFrame terá as colunas nomeadas de acordo com as características das flores, que são armazenadas em `Iris_dataset.feature_names`.

```
display(iris.head())
```

- O comando `display(iris.head())` exibe as primeiras linhas do DataFrame `iris`, mostrando uma amostra dos dados carregados.

```
# Construção do diagrama de dispersão
plt.scatter(iris['petal length (cm)'], iris['petal width (cm)'])
plt.scatter(iris['sepal length (cm)'], iris['sepal width (cm)'])
```

- Aqui, você está construindo um diagrama de dispersão (scatter plot) com base nos dados do conjunto Iris. Duas chamadas para `plt.scatter()` estão sendo usadas para plotar os comprimentos e larguras das pétalas e das sépalas das flores.

```
plt.xlabel('Comprimento [cm]')
plt.ylabel('Largura [cm]')
```

- Esses comandos definem os rótulos dos eixos x e y do gráfico.

```
plt.title('Conjunto de dados - Íris de Fisher')
```

- Este comando define o título do gráfico, indicando que se trata do Conjunto de Dados Iris de Fisher.

```
plt.legend(['Pétala', 'Sépala'])
```

- Adiciona uma legenda ao gráfico para indicar que as duas séries de dados representam as pétalas e as sépalas.

```
plt.grid()
```

- Adiciona uma grade ao gráfico para melhorar a legibilidade.

```
plt.show()
```

- Por fim, este comando exibe o gráfico de dispersão na tela.

O código carrega o conjunto de dados Iris de Fisher, cria um DataFrame para representar os dados e gera um diagrama de dispersão para visualizar as medidas de comprimento e largura das pétalas e sépalas das flores Iris. Isso é útil para a análise exploratória de dados e a visualização das características das flores

Dados emparelhados - Séries Temporais:

```
# Criação do conjunto de dados usando dicionários
dados = pd.DataFrame({'Assinantes': [134.6, 148.1, 169.5, 194.5, 219.7, 243.4,
                                     262.7, 276.6, 292.8, 306.3, 321.7],
                      'Conta média':[47.42, 49.46, 49.49, 49.52, 49.30, 49.94,
                                     48.54, 49.57, 47.47, 47.23, 47.16]})
dados.index = list(range(2002, 2013))
display(dados)

# Séries temporais
fig, assinantes = plt.subplots()
assinantes.plot(dados.index, dados['Assinantes'], 'bo-')
assinantes.set(xlabel='Ano', ylabel='Milhões de assinantes',
               title = 'Telefonia celular')
assinantes.tick_params(axis='y', colors='blue')
conta = assinantes.twinx()
conta.plot(dados.index, dados['Conta média'], 'go-')
conta.set(ylabel='Conta média [US$]')
conta.tick_params(axis='y', colors='green')
```

# Criação do conjunto de dados usando dicionários
dados = pd.DataFrame({'Assinantes': [134.6, 148.1, 169.5, 194.5, 219.7, 243.4,
262.7, 276.6, 292.8, 306.3, 321.7],
'Conta média':[47.42, 49.46, 49.49, 49.52, 49.30, 49.94,
48.54, 49.57, 47.47, 47.23, 47.16]})
dados.index = list(range(2002, 2013))

- Neste trecho, você está criando um DataFrame Pandas chamado `dados` usando um dicionário. O dicionário tem duas chaves ('Assinantes' e 'Conta média'), cada uma associada a uma lista de valores correspondentes. Esses valores representam o número de assinantes e a conta média em dólares ao longo de 11 anos. O índice do DataFrame é definido como uma lista de anos de 2002 a 2012.

display(dados)

- O comando `display(dados)` exibe o DataFrame `dados` na saída, mostrando as informações sobre o número de assinantes e a conta média ao longo dos anos.

Séries temporais
fig, assinantes = plt.subplots()

- Aqui, você está criando um gráfico de séries temporais usando Matplotlib. `fig, assinantes = plt.subplots()` cria uma figura e um objeto de eixo (`assinantes`) para o gráfico.

```
assinantes.plot(dados.index, dados['Assinantes'], 'bo-')
```

- Este comando gera o gráfico da série temporal para o número de assinantes. `dados.index` é usado no eixo x para representar os anos, `dados['Assinantes']` é usado no eixo y para representar o número de assinantes, e `'bo-'` define o estilo do gráfico como pontos azuis ('bo') conectados por linhas sólidas ('-').

```
assinantes.set(xlabel='Ano', ylabel='Milhões de assinantes',
           title='Telefonia celular')
```

- Estes comandos definem os rótulos dos eixos x e y, bem como o título do gráfico.

```
assinantes.tick_params(axis='y', colors='blue')
```

- Este comando ajusta os parâmetros dos marcadores no eixo y, definindo a cor como azul.

```
conta = assinantes.twinx()
```

- Aqui, você está criando um segundo eixo y, chamado `conta`, que compartilha o mesmo eixo x (`assinantes.twinx()`). Isso permitirá que você adicione uma segunda série ao gráfico com uma escala diferente.

```
conta.plot(dados.index, dados['Conta média'], 'go-')
```

- Este comando gera o gráfico da série temporal para a conta média em dólares. `dados.index` é usado novamente no eixo x para representar os anos, `dados['Conta média']` é usado no eixo y para representar a conta média, e `'go-'` define o estilo do gráfico como pontos verdes ('go') conectados por linhas sólidas ('-').

```
conta.set(ylabel='Conta média [US$]')
```

- Este comando define o rótulo do eixo y do segundo eixo (`conta`).

```
conta.tick_params(axis='y', colors='green')
```

- Este comando ajusta os parâmetros dos marcadores no eixo y do segundo eixo, definindo a cor como verde.

Em resumo, o código cria um gráfico de séries temporais que combina duas séries de dados: o número de assinantes de telefonia celular (representado em azul) e a conta média em dólares (representada em verde) ao longo de vários anos. Dois eixos y são usados para representar essas séries com escalas diferentes, permitindo a análise conjunta das tendências ao longo do tempo.