Guia Completo sobre Pandas 2

Este guia é uma continuação do material de estudo sobre a biblioteca Pandas. Na aula anterior, você já foi introduzido às estruturas de dados do Pandas, como Series e DataFrames, e aprendeu algumas operações essenciais de limpeza, filtragem e manipulação de dados. Agora, vamos aprofundar ainda mais nos recursos do Pandas e explorar suas aplicações no dia a dia.

1 Loc e iLoc

.loc

O método **.loc** é utilizado para localizar e acessar dados em seu DataFrame com base nos nomes das linhas e colunas. Ele permite recuperar os valores correspondentes às linhas e colunas especificadas.

Exemplo:

```
df.loc["nome_da_linha1", "nome_da_coluna1"]
```

Caso você não especifique uma coluna, o método retornará todas as colunas para a linha especificada.

Exemplo:

```
df.loc["nome_da_linha1", :]
```

É possível também selecionar várias linhas e colunas ao mesmo tempo.

Exemplo:

```
df.loc[["nome_da_linha1", "nome_da_linha2"], "nome_da_coluna1"]
```

Além disso, você pode nomear intervalos de linhas ou colunas.

Exemplo:

```
df.loc[["nome_da_linha1", "nome_da_linha2"], "nome_da_coluna1":]
```

.iloc

O método **.iloc** funciona de maneira semelhante ao **.loc**, mas em vez de usar nomes de linhas e colunas, ele usa índices numéricos.

Exemplo:

```
df.iloc[0, 0]
```

Você também pode obter o nome de uma linha ou coluna a partir do índice adicionando o **.name** no final.

Exemplo:

```
df.iloc[0].name
```

Assim como o **.loc**, o **.iloc** aceita sequências e intervalos de índices para selecionar linhas e colunas.

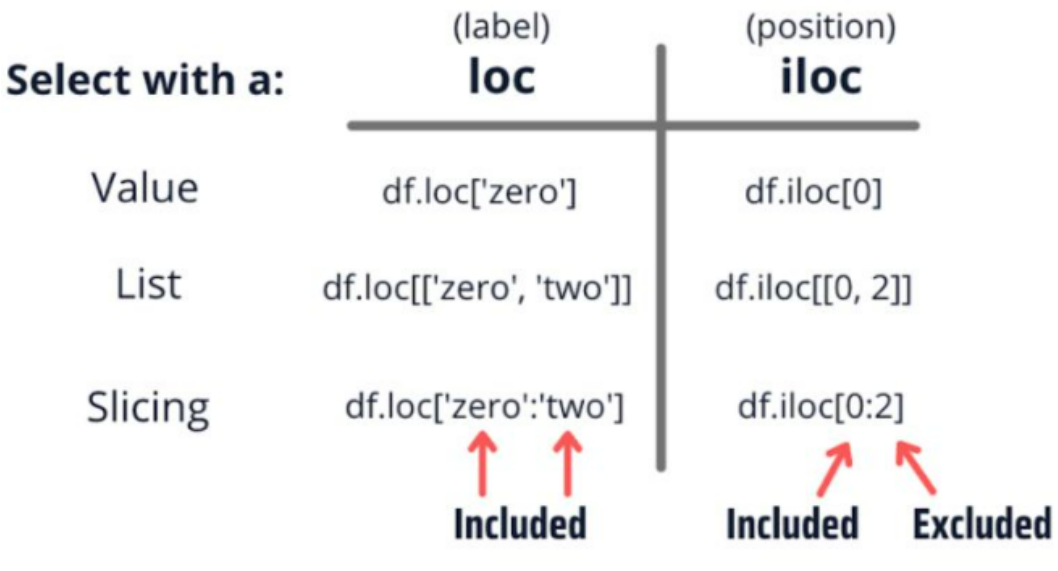
Exemplos:

```
df.iloc[[0, 5, 6]]
df.iloc[range(5), range(5)]
df.iloc[0:4]
```

Além disso, você pode usar o método **.unique()** após o **.loc** ou **.iloc** para retornar valores únicos, caso existam duplicatas.

Exemplo:

```
df.iloc[range(5)].unique()
```



Para ajudar a lembrar a diferença entre .loc e .iloc em Pandas, você pode usar o seguinte truque de memória:

.loc lembra de "localização"

- **.loc é usado para localizar os dados com base nos nomes das linhas e colunas.**
-

Portanto, .loc está associado à ideia de localizar dados com base em rótulos ou nomes de linhas e colunas.

.iloc lembra de "índices"

- **.iloc é usado para localizar os dados com base em índices numéricos das linhas e colunas.**
-

Portanto, .iloc está associado à ideia de localizar dados com base em índices numéricos, que são semelhantes às posições das linhas e colunas.

2. O que é uma função agregadora?

No Pandas, uma função agregadora é uma função que calcula um valor único a partir de um conjunto de valores em um DataFrame ou Série. Essas funções são frequentemente usadas em conjunto com o método **.groupby()** para agrupar dados com base em determinados critérios e, em seguida, calcular estatísticas para cada grupo. Isso nos ajuda a resumir informações de grandes conjuntos de dados de forma significativa.

As funções agregadoras nos ajudam a transformar dados em insights significativos e são particularmente úteis ao lidar com conjuntos de dados grandes ou na extração eficiente de informações.

3. Métodos Úteis

3.1 Groupby

O método **.groupby()** é usado para organizar um DataFrame em grupos com base nos valores de uma ou mais colunas. Isso permite aplicar funções agregadoras a cada grupo de dados.

Síntese Group By

```
df.groupby(by=coluna_de_agrupamento)
```

df: O DataFrame no qual você deseja realizar a operação **groupby**.
by: A coluna ou colunas que você deseja usar como critério de agrupamento.

Exemplo no Dia a Dia:

Imagine que você tem um conjunto de dados de vendas e deseja calcular a média de vendas por categoria de produtos. Você pode usar o **.groupby()** para agrupar os dados por categoria e, em seguida, calcular a média das vendas para cada categoria.

```
df.groupby("Categoria")["Vendas"].mean()
```

Neste exemplo, usamos **groupby('Categoria')** para agrupar os dados por categoria e, em seguida, aplicamos a função agregadora **.mean()** para calcular a média das vendas em cada categoria.

Ao usar o Groupby(), você geralmente aplica funções agregadoras para resumir os dados nos grupos resultantes.

- **.sum()**: Calcula a soma dos valores no grupo.
- **.mean()**: Calcula a média dos valores no grupo.
- **.median()**: Calcula a mediana dos valores no grupo.
- **.min()**: Encontra o valor mínimo no grupo.
- **.max()**: Encontra o valor máximo no grupo.
- **.count()**: Conta o número de valores não nulos no grupo.

3.2 Pivot Table

Uma tabela dinâmica (pivot table) é uma ferramenta que permite agrupar e reorganizar dados em formato de tabela. Ela nos ajuda a agregar, analisar e visualizar rapidamente dados de um DataFrame.

Exemplo no Dia a Dia:

Suponha que você tenha um grande conjunto de dados de vendas online e deseja criar um resumo que mostre a quantidade de vendas por mês e por categoria de produtos. Você pode usar uma pivot table para reorganizar os dados e obter uma visão clara das vendas mensais por categoria.

```
pivot = df.pivot_table(index="Mês", columns="Categoria", values="Quantidade Vendida", aggfunc="sum")
```

Aqui, a pivot table ajuda a resumir e visualizar os dados de forma eficiente.

3.3 Merge

O método **.merge()** é usado para combinar dois ou mais DataFrames com base em colunas ou índices comuns. Isso permite reunir dados de várias fontes em um único DataFrame.

Exemplo no Dia a Dia:

Suponha que você tenha um DataFrame com informações de funcionários e outro DataFrame com informações de departamentos. Você pode usar o **.merge()** para combinar esses DataFrames com base no ID do departamento, criando um único DataFrame com todas as informações relevantes.

Este é um exemplo prático em que o **.merge()** é usado para unir dados de diferentes fontes e facilitar a análise.

```
merged_df = pd.merge(df_funcionarios, df_departamentos, on="ID_Departamento")
```

3.4 Diff

A função **.diff()** é usada para calcular a diferença entre elementos consecutivos em uma Series ou DataFrame ao longo de um eixo. Isso é útil para analisar dados de séries temporais ou qualquer dado em que a diferença entre valores consecutivos seja importante.

Exemplo no Dia a Dia:

Imagine que você tenha um conjunto de dados de preços de ações e deseja calcular as variações diárias dos preços. Você pode usar o **.diff()** para calcular a diferença entre os preços de fechamento em dias consecutivos.

```
df["Variação Diária"] = df["Preço de Fechamento"].
```

Conclusão

Em resumo, o Pandas é uma biblioteca poderosa e versátil para manipulação e análise de dados. Neste guia, aprofundamos o conhecimento sobre o Pandas, cobrindo os métodos **.loc** e **.iloc** para seleção de dados, funções agregadoras para resumir informações em grupos usando **.groupby**, a criação de pivot tables para resumir dados de forma eficiente, o método **.merge** para combinar DataFrames, e as funções **.diff()**, **.pct_change()**, e **.shift()** para análise de séries temporais.

Essas ferramentas são essenciais para qualquer pessoa que trabalha com análise de dados, seja em finanças, ciência de dados, engenharia, ou qualquer outra área. Elas permitem realizar tarefas críticas, como resumir informações, calcular tendências, comparar dados e tomar decisões informadas.

À medida que você aprofunda seu conhecimento no Pandas e pratica esses conceitos com conjuntos de dados do mundo real, você estará melhor preparado para resolver problemas complexos e extrair informações valiosas de seus dados. O Pandas é uma habilidade essencial para qualquer profissional de análise de dados e um recurso valioso para tomar decisões baseadas em dados