

Começando com DataFrames:

Domine o básico em Python e R



Introdução:

Ao mexermos com tabelas de diferentes bancos de dados, muitas vezes, queremos extrair uma certa

Informação presente em um determinado índice.

Uma das diversas maneiras de fazermos essa Tarefa é utilizando linguagens de programação.

Para executar essa atividade, utilizamos, principalmente, as linguagens Python e R e suas bibliotecas Pandas e Dplyr, respectivamente.



SUMÁRIO:

- Importar e salvar
- Filtrar por colunas
- Filtrar por linhas
- Mostrar um dado específico
- Renomear colunas
- Remover valores nulos
- Alterar a classe da coluna
- Agrupar informações
- Obter valores em ordem
- Concatenar datasets
- Remover colunas
- Remover linhas

Importar e Salvar:



```
importar_exportar.py

1 # Importar:
2 df = pd.read_csv('arquivo.csv',
3                  sep = ";",
4                  encoding = "utf-8")
5
6 # Exportar:
7 df.to_csv('arquivo.csv',
8          index=False,
9          sep=';',
10         encoding='utf-8')
```



```
importar_exportar.R

1 # Importar:
2 df ← read.csv('arquivo.csv',
3              sep = ";",
4              encoding = "UTF-8")
5
6 # Exportar:
7 write.csv(df,
8          'arquivo.csv',
9          row.names = FALSE)
10
```

Parâmetros:

```
infos

1 # parâmetros comuns:
2 """
3 sep → as vezes, os numeros não inteiros podem
4 estar separando a fração do inteiro pela vírgula. Ex: 3,14.
5
6 encoding → é necessário usar essa função caso queira
7 usar caracteres não convencionais do inglês padrão.
8 """
9 # +parâmetros:
10 """
11 index → ao colocar como FALSE, significa que uma coluna
12 nova não será criada no arquivo novo apenas com os
13 números de índice.
14 """
15
```

Filtrar por colunas:

```
col_filter.py

1 # Selecionar colunas específicas
2 df[['coluna1', 'coluna2']]
3
```

```
col_filter.R

1 # Selecionar colunas específicas
2 df %>% select(coluna1, coluna2)
3
```



Filtrar por linhas:



```
row_filter.py

1 df[df['coluna'] > 10]
2 """ Filtra linhas onde a coluna
3 tem valores maiores que 10. """
4
```



```
row_filter.R

1 # Selecionar colunas específicas
2 df %>% select(coluna1, coluna2)
3
```

Parâmetros:

```
infos

1 # parâmetros
2 """
3 Python: Combine condições com & (AND) ou | (OR)
4 R: coluna %in% c(val1, val2) → filtra
5 valores específicos
6 """
7
```

Mostrar um dado específico:



```
select_data.py

1 df.loc[linha, 'coluna']
2 # Acessa pelo índice da linha e o nome da coluna.
3 df.iloc[linha, coluna]
4 # Acessa pela posição numérica.
5
```



```
select_data.R

1 df$coluna[linha]
2 # Acessa um elemento específico da coluna na linha.
3
```

Renomear colunas:



```
rename.py

1 # Renomear colunas
2 df.rename(columns={'coluna_antiga': 'coluna_nova'},
3            inplace=True # - sobrescreve o arquivo "df"
4            )
5
```



```
rename.R

1 # Renomear colunas
2 df %>% rename(coluna_nova = coluna_antiga)
3
```


Remover colunas e linhas:



```
remove.py

1 # Remover colunas
2 df.drop(['coluna1', 'coluna2'],
3         axis=1, # 1 → coluna
4         inplace=True)
5
6 # Remover linhas
7 df.drop([linha1, linha2],
8         axis=0, # 0 → linha
9         inplace=True)
10
```

```
remove.R

1 # Remover colunas
2 df <- df %>%
3   select(-coluna1, -coluna2)
4
5 # Remover linhas
6 df <- df[-c(linha1, linha2), ]
7
```



Obter valores em ordem



```
drop_na.py

1 # Ordenação:
2 df = df.sort_values('coluna', ascending= bool)
3 """
4 o ordem será crescente se ascending = True,
5 e decrescente se ascending = False
6 """
7
```



```
drop_na.R

1 #Ordenação:
2 df %>% arrange(coluna) # Crescente.
3 df %>% arrange(desc(coluna)) #Decrescente
4
```

Agrupar informações :



```
alter_type.py

1 # Agrupar por informações
2 df.groupby('coluna')['coluna2'].sum()
3
```



```
alter_type.R

1 # Agrupar por informações
2 df %>%
3   group_by(coluna) %>%
4   summarise(soma = sum(coluna2, na.rm = TRUE))
5
```

```

1 # parâmetros
2 """
3 Utilizado para aplicar funções, como sum(), mean(),
4 std() ou possíveis lambdas nos dados.
5 → Para calcular a quantidade dos valores agrupados,
6 usa-se .value_counts() em python e
7 summarise(count = n(), .groups = "drop") em R.
8 """
9

```

Remover valores nulos:

```

1 # Remover linhas com valores nulos
2 df.dropna(inplace=True)
3
4 # Remover colunas com valores nulos
5 df.dropna(axis=1, inplace=True)
6

```

```

1 # remove linhas:
2 df <- df %>% drop_na()
3 #remove colunas:
4 df <- df %>% select(where(function(coluna)
5                       !any(is.na(coluna))))
6

```



Alterar a classe da coluna:



```
alter_type.py

1 # Alterar a classe da coluna
2 df['coluna'] = df['coluna'].astype(type)
3 # → Não pode dar TypeError
4
```



```
alter_type.R

1 # # Alterar a classe da coluna
2 df <- df %>%
3   mutate(coluna = as.type(coluna))
4 # Trocar "type" pelo novo tipo da variável
5
```


Concatenar datasets:



```
alter_type.py

1 # Concatenar linhas
2 df_concat = pd.concat([df1, df2], axis=0)
3 # Concatenar colunas
4 df_concat = pd.concat([df1, df2], axis=1)
5
```



```
alter_type.R

1 # Concatenar linhas
2 df_concat ← bind_rows(df1, df2)
3 # Concatenar colunas
4 df_concat ← bind_cols(df1, df2)
5
```