Começando com DataFrames:

Domine o básico em Python e R

Matheus Felipe

Matheus Felipe

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

```python
# Importar:
df = pd.read_csv('arquivo.csv',
                 sep = ";",
                 encoding = "utf-8")

# Exportar:
df.to_csv('arquivo.csv',
          index=False,
          sep=';',
          encoding='utf-8')
```

importar_exportar.R

```r
# Importar:
df ← read.csv('arquivo.csv',
              sep = ";",
              encoding = "UTF-8")

# Exportar:
write.csv(df,
          'arquivo.csv',
          row.names = FALSE)

```

Parametros:

infos

```
# parâmetros comuns:
"""
sep → as vezes, os numeros não inteiros podem
estar separando a fração do inteiro pela vírgula. Ex: 3,14.

encoding → é necessário usar essa função caso queira
usar caracteres não convencionais do inglês padrão.
"""
# +parâmetros:
"""
index → ao colocar como FALSE, significa que uma coluna
nova não será criada no arquivo novo apenas com os
números de índice.
"""

```

Filtrar por colunas:

col_filter.py

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

```

col_filter.R

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

```

Filtrar por linhas:

row_filter.py

```python
df[df['coluna'] > 10]
""" Filtra linhas onde a coluna
tem valores maiores que 10. """

```

row_filter.R

```r
df %>%
  filter(coluna1 > 25, coluna2 = 'palavra')
```

Parametros:

infos

```
# parãmetros
"""
 Python: Combine condições com & (AND) ou | (OR)
 R: coluna %in% c(val1, val2) → filtra
    valores específicos
"""

```

Mostrar um dado específico:

select_data.py

```python
df.loc[linha, 'coluna']
# Acessa pelo índice da linha e o nome da coluna.
df.iloc[linha, coluna]
# Acessa pela posição numérica.

```

select_data.R

```r
df$coluna[linha]
# Acessa um elemento específico da coluna na linha.

```

Renomear colunas:

rename.py

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

```

rename.R

```r
# Renomear colunas
df %>% rename(coluna_nova = coluna_antiga)

```

Remover colunas e linhas:

remove.py

```python
# Remover colunas
df.drop(['coluna1', 'coluna2'],
        axis=1, # 1 → coluna
        inplace=True)

# Remover linhas
df.drop([linha1, linha2],
        axis=0, # 0 → linha
        inplace=True)

```

remove.R

```r
# Remover colunas
df ← df %>%
     select(-coluna1, -coluna2)

# Remover linhas
df ← df[-c(linha1, linha2), ]

```

Obter valores em ordem

drop_na.py

```python
# Ordenaação:
df = df.sort_values('coluna', ascending= bool)
"""
o ordem será crescente se ascending = True,
e decrescente se ascending = False
"""

```

drop_na.R

```r
#Ordenação:
df %>% arrange(coluna)  # Crescente.
df %>% arrange(desc(coluna)) #Decrescente

```

Agrupar informações :

alter_type.py

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

```

alter_type.R

```r
# Agrupar por informações
df %>%
    group_by(coluna) %>%
        summarise(soma = sum(coluna2, na.rm = TRUE))

```

infos

```
# parãmetros
"""
 Utilizado para aplicar funções, como sum(), mean(),
 std() ou possíveis lambdas nos dados.
→ Para calcular a quantidade dos valores agrupados,
usa-se .value_counts() em python e
summarise(count = n(), .groups = "drop") em R.
"""

```

Remover valores nulos:

drop_na.py

```
# Remover linhas com valores nulos
df.dropna(inplace=True)

# Remover colunas com valores nulos
df.dropna(axis=1, inplace=True)

```

drop_na.R

```
# remove linhas:
df ← df %>% drop_na()
#remove colunas:
df ← df %>% select(where(function(coluna)
                         !any(is.na(coluna))))

```

Alterar a classe da coluna:

alter_type.py

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

```

alter_type.R

```r
# # Alterar a classe da coluna
df ← df %>%
    mutate(coluna = as.type(coluna))
# Trocar "type" pelo novo tipo da variável

```

Concatenar datasets:

alter_type.py

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

```

alter_type.R

```r
# Concatenar linhas
df_concat ← bind_rows(df1, df2)
# Concatenar colunas
df_concat ← bind_cols(df1, df2)

```