home2

October 22, 2025

1 Bibliotecas

[1]:
```
import warnings
import os.path
import time
import psutil
import matplotlib.pyplot as plt
import plotly.express    as px
import plotly.io         as pio
import pandas            as pd
import numpy             as np
from   pyexcelerate  import Workbook
```

[ ]:
```
```

2 Definições Iniciais

[2]:
```
warnings.filterwarnings('ignore', category=UserWarning, module='openpyxl')
# define pasta de trabalho com os dados e nome do arquivo original a ser lido
base_path = os.path.abspath(os.getcwd()) + '/dataset/'
in_file = 'trackLogcivic.xlsx'
```

[3]:
```
# Cria os dataframes de comparativo de tempo
df_read = pd.DataFrame({'Read_engine': [np.nan],'Read_time[s]': [np.nan]})
df_save = pd.DataFrame({'Save_engine': [np.nan],'Save_time[s]': [np.nan],␣
 ↪'File_size': [np.nan]})
df_save_parquet = pd.DataFrame({'Save_engine': [np.nan],'Save_time[s]': [np.
 ↪nan], 'File_size': [np.nan]})

df_read = df_read.astype(str)
df_save = df_save.astype(str)
df_save_parquet = df_save_parquet.astype(str)
```

3 Funções

[4]:
```
def result_graf(x_label, y_label, graph_label, max_width, max_height, df_aux,␣
↪y_col_number):
    # fig = px.bar(df_aux, y= 'Read_time[s]', x='Read_engine',␣
↪title=graph_label, width=max_width, height=max_height)
    fig = px.bar(x= df_aux.iloc[:,0], y=df_aux.iloc[:,y_col_number],␣
↪title=graph_label, width=max_width, height=max_height, text=df_aux.iloc[:
↪,y_col_number])
    fig.update_layout(title_font = {"size": 22},
                      font_family="Arial",
                      font_color="Black",
                      title_font_family="Arial",
                      title_font_color="black",
                      title_x=0.15,)
                      # xaxis={'categoryorder': 'category ascending'})
    # configura os títulos dos eixos
    fig.update_xaxes(tickangle = 0,
                     title_text = x_label,
                     title_font = {"size": 18},
                     tickfont_size=14)

    fig.update_yaxes(title_text = y_label,
                     title_font = {"size": 18},
                     tickfont_size=14)
    return fig
```

[5]:
```
# def result_graf(x_label, y_label, graph_label, max_width, max_height, df_aux,␣
↪y_col_number):
#     # Cria a figura e o eixo
#     fig, ax = plt.subplots(figsize=(max_width/100, max_height/100))

#     # Cria o gráfico de barras usando Matplotlib
#     ax.bar(df_aux.iloc[:, 0], df_aux.iloc[:, y_col_number], label=graph_label)

#     # Configura o título e os rótulos dos eixos
#     ax.set_title(graph_label, fontsize=22, fontfamily='Arial', color='black',␣
↪loc='left', pad=20)
#     ax.set_xlabel(x_label, fontsize=18, fontfamily='Arial')
#     ax.set_ylabel(y_label, fontsize=18, fontfamily='Arial')

#     # Configura o tamanho das fontes dos ticks
#     ax.tick_params(axis='x', labelsize=14)
#     ax.tick_params(axis='y', labelsize=14)

#     # Adiciona os valores das barras como texto acima de cada barra
#     for i in ax.containers:
```

```
#         ax.bar_label(i, label_type='edge', fontsize=12, padding=3)

#     # Ajusta o layout
#     plt.tight_layout()

#     # Retorna a figura
#     return fig
```

4 Leitura via pandas usando diferentes engines

[6]:
```
start_exec_time = time.time()
df = pd.read_excel(base_path + in_file, dtype=str)
end_exec_time = time.time()
print("Excel Read File - no engine", round(end_exec_time-start_exec_time,1) ,␣
 ↪"s")
df_read.loc[len(df_read),:] = ['Nenhum', round(end_exec_time-start_exec_time,1)]
```

```
Excel Read File - no engine 127.1 s
```

[7]:
```
start_exec_time = time.time()
df = pd.read_excel(base_path + in_file, dtype=str, engine='openpyxl')
end_exec_time = time.time()
print("Excel Read File - openpyxl engine",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
df_read.loc[len(df_read),:] = ['openpyxl',␣
 ↪round(end_exec_time-start_exec_time,1)]
```

```
Excel Read File - openpyxl engine 193.5 s
```

[8]:
```
start_exec_time = time.time()
df = pd.read_excel(base_path + in_file, dtype=str, engine='calamine')
end_exec_time = time.time()
print("Excel Read File - calamine engine",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
df_read.loc[len(df_read),:] = ['calamine',␣
 ↪round(end_exec_time-start_exec_time,1)]
```

```
Excel Read File - calamine engine 33.6 s
```

[9]:
```
in_file = 'parquet_auto.xlsx'
start_exec_time = time.time()
df = pd.read_parquet(base_path + in_file)
end_exec_time = time.time()
print("Excel Read File - parquet format",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
df_read.loc[len(df_read),:] = ['parquet',␣
 ↪round(end_exec_time-start_exec_time,1)]
```

```
Excel Read File - parquet format 3.6 s
```

5 Salva por diferentes métodos

5.1 Salva em formato XLSX

5.1.1 Via Pandas, diferentes engines

[10]:
```
out_file = 'none.xlsx'
start_exec_time = time.time()
df.to_excel(base_path + out_file, sheet_name= out_file[0:-5], index=False)
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - no engine", round(end_exec_time-start_exec_time,1) ,␣
 ↪"s")
print(f'Arquivo com {file_size} bytes')
df_save.loc[len(df_save),:] = ['no engine',␣
 ↪round(end_exec_time-start_exec_time,1), file_size]
```

```
Excel Save File - no engine 246.4 s
Arquivo com 61482449 bytes
```

[11]:
```
out_file = 'xlsxwriter.xlsx'
start_exec_time = time.time()
df.to_excel(base_path + out_file, sheet_name= out_file[0:-5],␣
 ↪engine='xlsxwriter', index=False)
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - xlsxwriter engine",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
print(f'Arquivo com {file_size} bytes')
df_save.loc[len(df_save),:] = ['xlsxwriter engine',␣
 ↪round(end_exec_time-start_exec_time,1), file_size]
```

```
Excel Save File - xlsxwriter engine 247.2 s
Arquivo com 61482456 bytes
```

[12]:
```
out_file = 'openpyxl.xlsx'
start_exec_time = time.time()
df.to_excel(base_path + out_file, sheet_name= out_file[0:-5],␣
 ↪engine='openpyxl', index=False)
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - openpyxl engine",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
print(f'Arquivo com {file_size} bytes')
df_save.loc[len(df_save),:] = ['openpyxl engine',␣
 ↪round(end_exec_time-start_exec_time,1), file_size]
```

```
Excel Save File - openpyxl engine 320.5 s
Arquivo com 65136372 bytes
```

5.1.2 Via pyexcelerate

[13]:
```
out_file = 'pyexcelerate.xlsx'
start_exec_time = time.time()
df = df.fillna('')
data = [df.columns.tolist(), ] + df.values.tolist()
wb = Workbook()
wb.new_sheet(out_file[0:-5], data = data)
wb.save(base_path + out_file)
del data
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - pyexcelerate library",␣
 ↪round(end_exec_time-start_exec_time,1) , "s")
print(f'Arquivo com {file_size} bytes')
df_save.loc[len(df_save),:] = ['pyexcelerate library',␣
 ↪round(end_exec_time-start_exec_time,1), file_size]
```

```
Excel Save File - pyexcelerate library 108.1 s
Arquivo com 68186450 bytes
```

5.2 Salva em formato Parquet

[14]:
```
out_file = 'parquet_auto.parquet'
start_exec_time = time.time()
df.to_parquet(base_path + out_file)
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - parquet format auto engine",␣
 ↪round(end_exec_time-start_exec_time,2) , "s")
print(f'Arquivo com {file_size} bytes')
df_save_parquet.loc[len(df_save_parquet),:] = ['parquet_auto engine',␣
 ↪round(end_exec_time-start_exec_time,2), file_size]
```

```
Excel Save File - parquet format auto engine 2.55 s
Arquivo com 35213960 bytes
```

[15]:
```
out_file = 'parquet_fastparquet.parquet'
start_exec_time = time.time()
df.to_parquet(base_path + out_file, engine='fastparquet')
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - parquet format fastparquet engine",␣
 ↪round(end_exec_time-start_exec_time,2) , "s")
print(f'Arquivo com {file_size} bytes')
```

```
df_save_parquet.loc[len(df_save_parquet),:] = ['parquet_fastparquet engine',␣
 ↪round(end_exec_time-start_exec_time,2), file_size]
```

```
Excel Save File - parquet format fastparquet engine 3.18 s
Arquivo com 46274285 bytes
```

[16]:
```
out_file = 'parquet_pyarrow.parquet'
start_exec_time = time.time()
df.to_parquet(base_path + out_file, engine='pyarrow')
end_exec_time = time.time()
file_size = os.path.getsize(base_path + out_file)
print("Excel Save File - parquet format pyarrow engine",␣
 ↪round(end_exec_time-start_exec_time,2) , "s")
print(f'Arquivo com {file_size} bytes')
df_save_parquet.loc[len(df_save_parquet),:] = ['parquet_pyarrow engine',␣
 ↪round(end_exec_time-start_exec_time,2), file_size]
```

```
Excel Save File - parquet format pyarrow engine 2.5 s
Arquivo com 35213960 bytes
```

6 Organização dos Resultados

[17]:
```
# faz conversão de tipo para os dados numéricos
df_read['Read_time[s]'] = df_read['Read_time[s]'].astype(float)
df_read.dropna(inplace=True)
df_read = df_read.sort_values(by = 'Read_time[s]', ascending = True).
 ↪reset_index(drop=True)
```

[18]:
```
filtro = df_save.Save_engine != 'nan'
df_save = df_save.loc[filtro,:]
# faz conversão de tipo para os dados numéricos
df_save['Save_time[s]'] = df_save['Save_time[s]'].astype(float)
df_save['File_size'] = df_save['File_size'].astype('Int64')
df_save = df_save.sort_values(by = 'Save_time[s]', ascending = True).
 ↪reset_index(drop=True)
```

[19]:
```
filtro = df_save_parquet.Save_engine != 'nan'
df_save_parquet = df_save_parquet.loc[filtro,:]
# faz conversão de tipo para os dados numéricos
df_save_parquet['Save_time[s]'] = df_save_parquet['Save_time[s]'].astype(float)
df_save_parquet['File_size'] = df_save_parquet['File_size'].astype('Int64')
df_save_parquet = df_save_parquet.sort_values(by = ['Save_time[s]',␣
 ↪'File_size'], ascending = True).reset_index(drop=True)
```

[20]:
```
# junta dados de gravação
df_save_mixed = pd.concat([df_save, df_save_parquet])
```

```
df_save_mixed = df_save_mixed.sort_values(by = ['Save_time[s]', 'File_size'],␣
 ↪ascending = True).reset_index(drop=True)
```

[21]:
```
# df_save_mixed.loc[filtro,:]
```

[ ]:
```

```

7 Resultados

7.1 Leitura dos dados XLSX:

- O arquivo de entrada é uma tabela Excel (xlsx) com ~73MB formado por 22 colunas e 375.617 linhas.
- Houve uma variação de ~6x entre o método de leitura mais rápido e o mais lento, sendo o engine 'calamine' o mais veloz para coleta dos dados em xlsx (figura 1);

7.2 Gravação de dados XLSX:

- O xlsxwriter engine e não utilizar engine algum tem praticamente o mesmo resultado em tempo de gravação;
- O openpyxl foi de 30% mais lento para transcrever o dataframe em um arquivo xlsx;
- Já a biblioteca pyexcelerate (pip install PyExcelerate) foi supreendentemente 3x mais veloz; em contrapartida:
  - o tamanho do arquivo foi 10% maior;
  - os dados necessariamente devem ser salvos em formato texto;
  - o dataframe é duplicado em memória no processo de gravação dos dados, podendo ser um problema em casos de dataframes muito grandes;

7.3 Alternativa:

- em casos em que o mesmo arquivo será aberto e salvo algumas vezes, pode ser interessante mudar o formato dos dados de xlsx para parquet:
- não existe variação significativa entre os diferentes engines de gravação em formato parquet. Todos tiveram tempos de execução:
  - ~40x mais rápido que o pyexcelerate;
  - ~100x mais rápido que o pandas de melhor performance;
- Já o tempo de leitura ao leitura de dados em formato parquet, foi cerca de 22x mais rápido que o melhor resultado obtido na leitura do arquivo excel (figura 5).

[22]:
```
filtro = df_read.Read_engine!='parquet'
fig = result_graf('Motor de leitura (engine)', 'Tempo de leitura dos dados [s]',
            'Figura 1: Velocidade de Leitura Arquivo xlsx via Pandas', 800,␣
 ↪400, df_read.loc[filtro,:], 1)
fig.show()
# fig.savefig("figura 1.png")
```

[23]:
```
# fig.write_image("fig1.jpeg")
```

[24]:
```
fig = result_graf('Motor de leitura (engine)', 'Tempo de gravação dos dados␣
 ↪[s]',
            'Figura 2: Velocidade de Gravação Arquivo xlsx', 800, 400, df_save,␣
 ↪1)
fig.show()
# fig.savefig("figura 2.png")
```

[25]:
```
fig = result_graf('Motor de leitura (engine)', 'Tempo de leitura dos dados [s]',
                  'Figura 3: Tamanho do Arquivo xlsx Gerado', 800, 400, df_save.
 ↪sort_values(by='File_size'), 2)
fig.show()
# fig.savefig("figura 3.png")
```

[26]:
```
fig = result_graf('Motor de leitura (engine)', 'Tempo de leitura dos dados [s]',
            'Figura 4: Velocidade de Gravação do em Formato PARQUET', 800, 400,␣
 ↪df_save_parquet,1)
fig.show()
# fig.savefig("figura 4.png")
```

[27]:
```
fig = result_graf('Motor de leitura (engine)', 'Tempo de leitura dos dados [s]',
            'Figura 5: Velocidade de Leitura Arquivo XLSX vs PARQUET', 800,␣
 ↪400, df_read.head(2),1)
fig.show()
# fig.savefig("figura 5.png")
```

[28]:
```
filtro = (df_save_mixed.Save_engine=='pyexcelerate library') | (
          df_save_mixed.Save_engine=='openpyxl engine') | (
          df_save_mixed.Save_engine=='parquet_fastparquet engine') | (
          df_save_mixed.Save_engine=='parquet_pyarrow engine')

fig = result_graf('Entre 89x e 22,7x mais rápido gravar em parquet', 'Tempo de␣
 ↪gravação dos dados [s]',
            'Figura 6: Velocidade de Gravação Arquivo XLSX x PARQUET', 900,␣
 ↪400, df_save_mixed.loc[filtro,:], 1)
fig.show()
# fig.savefig("figura 6.png")
```

8 Conclusões

Quando os dados estão disponíveis em formato Excel, o tempo de leitura e gravação desses dados pode ser bastante significativo. Ainda na etapa de desenvolvimento, o impacto na produtividade pode penalizar o projeto. Nesses casos, pode ser interessante: - Migrar os dados para o formato parquet e durante todo o desenvolvimento utilizar esse formato para leitura e gravação dos dados; - A versão a ser colocada em produção pode voltar ao formato xlsx original, caso seja necessário; - Caso existam etapas intermediárias onde o pré-processamento (ETL, por exemplo) precisa ser salvo, o impacto do uso do formato parquet pode ser ainda maior.

Estudo de caso: Um conjunto de arquivos com dados de um processo produtivo agrícola precisam ser abertos, pré-processados (ETL) e salvos para uso futuro. Posteriormente, os dados processados e vinculados gerando o resultado final que será salvo em formato Excel.

- Dados de entrada: 9 arquivos de entrada todos eles em formato xlsx, sendo:
  - o menor arquivo: ~9MB;
  - o maior arquivo: ~40MB;
  - tamanho total: ~200MB;
- Arquivos em XLSX:
  - Ciclo de leitura, processamento e gravação dos resultados: ~700s
- Resultado da migração para Parquet:
- Arquivos em Parquet:
  - Dados de entrada;
  - Arquivos intermediários de pré-processamento;
- Arquivo mantido em XLSX:
  - Arquivos de saída com resultado final do processamento:
    * ~65MB
    * ~17MB
  - Ciclo de leitura, processamento e gravação dos resultados: ~200s

[ ]: