Esse Documento tem como base analisar meu documento criado(Com base no Desafio de número 6, da Fábrica de Software(“Análise de Dados”)

* Pretendo, mais uma vez, ser bastante objetivo, utilizando de termos e expressões recorrentes da área, pois a correção será realizada por um corpo competente;
* Irei colocar o passo a passo dos Dados(Em Linguagem Phyton), explicando cada passo e o que aquele comando vai fazer(sua execução)

Célula 1 (índice 0)Célula 1 (índice 0)

Parte 1 da célula 1Parte 1 da célula 1

* import pandas as pd  
  df = pd.read\_csv('/content/dados\_nao\_tratados.csv')  
  display(df)

-Explicação  
 -,eu importei a biblioteca pandas, visando poder manipular dados em formato de tabela.

Em seguida, utilizei a função pd.read\_csv(), solicitada pelo instrutor da aula, para carregar o arquivo dados\_nao\_tratados.csv.

- Variáveis criadas/alteradas: df.

- Eu usei df para armazenar um resultado intermediário ou final. -

- Entradas: um arquivo CSV foi lido. Eu usei esse arquivo como fonte de dados brutos.-

- Saídas: o resultado foi colocado em uma variável (DataFrame) para ser manipulado depois.-

Reesultado dessas escolhas: a tabela carregada agora está disponível na memória

Célula 2 (índice 1)

Parte 1 da célula 2

* df['idade'] = pd.to\_numeric(df['idade'], errors='coerce')  
  df["idade"]=df["idade"].astype('Int64')

display(df)

explicando um pouco mais sobre esse trecho

- eu converti a coluna idade para o formato numérico utilizando pd.to\_numeric().

-Usei o parâmetro errors='coerce' para que qualquer valor inválido fosse transformado em nulo (NaN), na tentativa de garantir uma maior consistência

- Em seguida, converti a coluna para o tipo Int64(já que a idade não pode ser um valor “quebrado”), que permite trabalhar com números inteiros mas ainda aceita valores ausentes.

- Isso foi necessário para padronizar a coluna e permitir operações matemáticas sem erros.( tentei mudar diretamente antes de anular os erros com essa função, e sempre respaldava em outro tipo de erro, então achei melhor zerar par depois fazer as alterações)

- Variáveis criadas/alteradas: errors.

- Eu usei erros para armazenar um resultado intermediário ou final.

Parte 2 da célula 2

-essa parte irei explicar apenas uma vez, por mais que eu tenha utilizado várias vezes, para não ficar um documento muito massivo para o CO Wallace( ou para qualquer outro competente que esteja avaliando o trabalho)

* display(df)

- sempre que eu utilzei o display(df) a ideia era conferir visualmente as alterações feitas em cada etapa.

Célula 4 (índice 3)

Parte 1 da célula 4

* df["ativo"]=df["ativo"].astype(str).str.strip().str.lower().map({  
   "sim":True,  
   "não":False,  
   "true":True,  
   "false":False,  
   "yes":True,  
   "nao":False,  
   "nan":False,  
   "TRUE":True,  
   "FALSE:":False,  
   })
* display(df)df["ativo"]=df["ativo"].astype(str).str.strip().str.lower().map({ "sim":True, "não":False, "true":True, "false":False, "yes":True, "nao":False, "nan":False, "TRUE":True, "FALSE:":False, })display(df)

-Considerei a parte mais complexa do meu código, mas aqui vai a explicação do pq esse formato foi utilizado

--, eu normalizei os valores da coluna ativo.

- Primeiro converti tudo para string, (removi espaços em branco e padronizei para minúsculas, deixar a tabela mais uniforme, dando um maior profissionalismo na ideia).

-Em seguida, utilizei um mapeamento (map) para transformar diferentes formas de escrita (‘sim’, ‘yes’, ‘true’, ‘não’, ‘nao’, etc....abarcando todas as possibilidades possíveis)

- em valores booleanos (True ou False). Dessa forma, a coluna passou a ter apenas duas categorias padronizadas, eliminando ambiguidades

Célula 5 (índice 4)Célula 5 (índice 4)

Parte 1 da célula 5Parte 1 da célula 5

* df['data\_inscricao']=pd.to\_datetime(df['data\_inscricao'], errors='coerce')  
  display(df)df
* -essa foi a parte que deu uma maior dor de cabeça, pois o código

-Nessa parte eu usei esse código para deixar nulo os meus campos que estavam apresentando uma certa “distorção” dos padrões analisados na tabela

-- Eu converti a coluna data\_inscricao para o tipo datetime (porque precisava trabalhar com datas de forma correta.) Muitas vezes os dados vêm como texto, e assim não dá para fazer operações com datas corretamente.

- Usei errors='coerce' para que qualquer valor que não fosse uma data válida se tornasse nulo(evitando que o código quebre e permitindo tratar esses casos depois.)

Célula 6 (índice 5)

Parte 1 da célula 6

* df['ativo']=df['ativo'].fillna("sem cadastro")  
  display(df)

-aqui merece uma ideia mais detalhada do por que essa ideia foi utilizada

--eu tratei os valores ausentes da coluna ativo. Sempre que não havia cadastro, eu substituí por uma string padrão "sem cadastro". Isso garante que a tabela não tenha valores nulos e melhora a interpretação dos dados.

-

Célula 7 (índice 6)

Parte 1 da célula 7

* df=df.fillna(0)  
  display(df)

Explicação de pq esse código foi utilizado

- pensei em tratar todos os valores ausentes do DataFrame substituindo-os por zero com a função fillna(0). Pensei em evitar problemas em cálculos matemáticos, já que valores nulosnão faziam muito sentido na atividade proposta.

E aqui finalizamos a apresentação das ideias usadas nesse Desafio.

* Espero que esse documento tenha sido esclarecedor e objetivo;
* Tentei deixar essa documentação mais pessoal, para que o competente da área tenha entendido o porque de cada coisa e cada lugar, mas sem perder a profissionalidade do documento;

-