DOCUMENTAÇÃO

TRATAMENTO DATASET: ‘CASOS COVID’

1- Importação da biblioteca ‘Pandas’ e ‘Numpy’:

Para que possamos utilizar as funcionalidades de cada uma das bibliotecas no Google Colab, é necessário que inicialmente façamos a importação.

2- Integração com a Google Cloud Platform (GCP):

Cumprindo o requisito de armazenar os datasets originais no ‘bucket’ da GCP, esse passo é essencial para que possamos conectar a sessão do Colab com a conta administradora da GCP, fornecendo, assim, permissão para que possamos importar para o ‘notebook’ o dataset desejado.

3- Importação do Dataset por meio do comando ‘gsutil’:

O ‘gsutil’ é uma aplicação em Python que permite acesso a GCP pela linha de comando. Neste passo é utilizada em conjunto com o comando ‘cp’ para copiar o arquivo desejado no ‘bucket’ para uma pasta temporária (tmp) nas dependências do Colab. A seguir, por meio da função ‘pd.read’, fazemos a leitura do dataset localizado na pasta tmp, obtendo nosso objeto de trabalho: ‘dataframe’ (df).

4- Verificação dos tipos de dados:

Parte fundamental do tratamento dos dados envolve a disposição da informação correta, mas, também, no formato correto - do contrário, algumas manipulações podem ficar indisponíveis/incorretas (por exemplo, ao somarmos ‘strings’ o resultado é a concatenação dos valores, não propriamente a soma). Portanto, nesse passo, usamos a funcionalidade ‘df.dtypes’ para verificarmos qual o formato dos dados contidos no dataframe.

5- Backup do Dataset:

Etapa de segurança: caso seja feita alguma mudança equivocada sobrescrevendo o dataframe, para que não precisemos reimportá-lo, é feito um backup por meio do comando ‘df.copy’.

6- Tradução dos títulos das colunas:

Visto a obrigatoriedade das informações estarem em pt-br, nesta etapa fazemos a tradução dos títulos das colunas para o idioma adequado. Para isso, utilizamos o comando ‘df.rename’ (para renomear) em conjunto com o ‘inplace = true’ (para fixar a alteração no dataframe)

7- Verificando o sucesso da etapa anterior:

Para verificarmos a alteração feita no passo anterior, utilizamos o comando 'df.head', para nos mostrar apenas as cinco primeiras linhas do dataframe. O motivo de termos utilizado o 'df.head' ao invés do 'df' se deve ao fato de que os valores a serem verificados eram os nomes das colunas, não havendo necessidade de gastarmos mais processamento da máquina exibindo além das cinco linhas iniciais.

8- Análises das colunas:

a) 'País':

Por meio do comando 'pd.unique', analisamos a coluna 'País' para identificarmos quais valores únicos estavam presentes. Essa análise foi crucial para que tomássemos a decisão de 'dropar' (eliminar) a coluna dos países, visto que ela só nos retornava valores = 'Brasil' - informação que, pelo escopo do trabalho, já era evidente. A coluna será 'dropada' na etapa 12.

b) 'Cidade':

Por meio do comando 'pd.unique', analisamos a coluna 'Cidade' para identificarmos quais valores únicos estavam presentes. Por meio dela, identificamos que só havia o valor 'TOTAL'. O motivo desse retorno é os dados estarem nos mostrando que as informações são relativas ao total das cidades no estado em questão - detalhe irrelevante para o escopo do trabalho, portanto, a coluna será 'dropada' na etapa 12.

c) 'Estado':

Por meio do comando 'pd.unique', analisamos a coluna 'Estado' para identificarmos quais valores únicos estavam presentes. Conseguimos, dessa forma, perceber a adição de 'TOTAL' à lista de valores únicos (que somente deveria conter os estados do Brasil). Esse novo 'TOTAL' contemplava os valores de todas as UFs compiladas - informação não relevante para o teor das análises futuras - portanto, linhas que continham esse valor na coluna 'Estado' serão 'dropadas' na etapa a seguir.

9- Eliminando linhas que continham valor 'Estado' igual a 'TOTAL':

De acordo com o racional explicitado na etapa 8-C, fazemos, então, uso do comando 'df.drop' (eliminar) em conjunto com o comando 'df.loc' (localizar) para que pudéssemos localizar, por meio da equivalência 'Estado == TOTAL' , e, juntamente, eliminar as linhas indesejáveis.

10- Verificando o sucesso da etapa anterior:

Para verificarmos a alteração feita na etapa anterior, utilizamos novamente o comando 'pd.unique' para visualizar os valores únicos contidos na coluna 'Estado'. Sucesso: 'TOTAL' fora removido.

11- Verificando o número de estados:

Apesar de já termos verificado que não existiam valores discrepantes na coluna dos estados, optamos por sermos extra cautelosos e rodar mais uma verificação da quantidade de valores únicos. Como temos 27 unidades federativas, a quantidade deveria ser exatamente 27. O funcionamento é: (para cada) + (valor único) + (somar 1 ao contador). A transcrição para código: (for) + (pd.unique) + (cont += 1).

12- Eliminando colunas sem pertinência para o estudo:

Visto que o dataset em análise já é o terceiro a ser utilizado no trabalho, nessa etapa tomamos a liberdade de reduzir os dados ao estritamente necessário para o desenvolvimento das análises e correlações pretendidas.

13- Verificando o sucesso da etapa anterior:

Com um simples comando 'df' visualizamos as 5 primeiras e 5 últimas linhas do dataframe. O suficiente para confirmarmos o sucesso do 'drop'.

14- Eliminando possíveis duplicatas:

Nessa etapa utilizamos o comando 'df.drop_duplicates' para que, na existência de valores duplicados, as cópias fossem eliminadas. Felizmente, devido a qualidade da fonte dos dados, não haviam duplicatas.

15- Verificando a existência de dados nulos:

Para estudarmos a quantidade de nulos no df, utilizamos os comandos 'print' + 'df.isnull' + 'sum', para que, no caso de dados nulos, esses fossem somados e a soma exibida na saída da célula. Felizmente, sem dados faltantes.

16- Convertendo o tipo dos dados em 'Data' e 'Estado':

Com o intuito de facilitar futuras operações neste dataset, optamos por converter o tipo de dado (dt type) da coluna 'Data' para 'datetime' (o formato correto para manipulação de datas), juntamente com a conversão do 'dt type' da coluna 'Estado' para string (texto) ao invés de 'object'.

17- Exportando o Dataframe tratado para o 'bucket' na GCP:

Seguindo a proposta do trabalho, realizamos o salvamento diretamente no 'bucket'. Para tanto, utilizamos primeiramente o comando 'df.to_csv' para criar um novo arquivo csv a partir do dataframe tratado. A seguir, assim como na importação, usamos o 'gsutil cp' a fim de copiar o arquivo pronto, presente nas dependências do Colab, para o Cloud Storage. Ao término do comando de cópia, temos o último comando 'gsutil cat', que é posto para que recebamos uma confirmação do sucesso da operação.