DOCUMENTAÇÃO

TRATAMENTO DATASET: ‘PEDÁGIO’

1- Importação da biblioteca ‘Pandas’ e ‘Numpy’ + função ‘json_normalize’ :

Para que possamos utilizar as funcionalidades de cada uma das bibliotecas no Google Colab, é necessário que inicialmente façamos a importação. Paralelamente, a função ‘normalize’ desempenha papel crucial na extração correta do arquivo em json para o Pandas.

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’ os datasets desejados.

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 os arquivos desejados no ‘bucket’ para uma pasta temporária (tmp) nas dependências do Colab.

4- Fazendo a leitura e codificação dos dados presentes na pasta ‘tmp’:

Apesar de termos explorado uma nova forma para a leitura dos dados, acabamos, ao fim, retornando ao modelo tradicional ‘pd.read’ para que pudéssemos ler os dados em dataframes.

5- Exibindo o Dataframe:

Para que verificarmos se a importação do arquivo json foi feita de forma correta, utilizamos o comando ‘df’ para exibir, muito resumidamente, as informações necessárias para validar o sucesso da operação.

6- Normalizando os dados contidos nos arquivos JSON

A função que importamos, ‘json_normalize’, agora desempenha, enfim, seu papel em normalizar os dados contidos no formato de lista do json, para que, em Pandas, haja a correta distribuição dos valores da lista no dataframe.

7- 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.info' para verificarmos qual o formato dos dados contidos no dataframe.

8 e 9- Convertendo o tipo dos dados de 'mes_ano' no Dataframe 1 e 2:

Com o intuito de facilitar futuras operações nesses dfs, optamos por converter o tipo de dado (dt type) das coluna 'mes_ano' de ambos os dataframes para 'datetime' (o formato correto para manipulação de datas).

10- Concatenando Dataframes:

Para aumentar o escopo da análise, optamos pela utilização de dois datasets (2019-2020). Portanto, no sentido de torná-los um - facilitando os tratamentos - fizemos uma lista contendo ambos dfs, seguida pelo comando 'pd.concat'.

11- 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 'concat'.

12- Verificação dos tipos de dados:

Repetindo a lógica do passo 7, novamente, executamos uma verificação do tipo de dados após a concatenação dos dataframes.

13- Alterando o tipo dos dados:

De acordo com a lógica estabelecida na etapa 7, agora, tomamos a decisão de alterar o tipo dos dados. Cada qual para o formato ideal tendo em vista as manipulações e análises que serão feitas futuramente.

14- Verificando o sucesso da etapa anterior:

Usando o comando 'df.info' visualizamos os tipos de dados do dataframe. Exatamente o que precisamos para confirmarmos o sucesso da conversão realizada na etapa 13.

15- Convertendo o valor da coluna 'Volume_Total' de 'object' para 'numeric':

Visando facilitar manipulações futuras, alteramos o tipo de dado dos valores em 'Volume_Total' para numérico.

16- Verificando o sucesso da etapa anterior:

Usando o comando 'df.info' visualizamos os tipos de dados do dataframe. Exatamente o que precisamos para confirmarmos o sucesso da conversão realizada na etapa 15.

17- 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'.

18- Visualizando o Dataframe

19- Renomeando 'mes_ano' para 'Data':

Para melhorar a objetividade e clareza dos dados, optamos por renomear a coluna 'mes_ano' para 'Data'.

20 e 21- Análise das colunas:

a) 'Sentido': *Por meio do comando 'pd.unique' verificamos a existência ou não de dados pertinentes. Feedback negativo, logo, a coluna será futuramente 'dropada'.*

b) 'Categoria': *Por meio do comando 'pd.unique' verificamos a existência ou não de dados pertinentes. Feedback negativo, logo, a coluna será futuramente 'dropada'.*

22- Eliminando coluna 'Categoria' + backup 'Categoria':

Apesar de, por hora, termos decidido 'dropar' a coluna 'Categoria' - seguindo o racional da etapa 21-B - paralelamente, optamos por fazer um backup em caso de mudarmos de ideia.

23- Visualizando o Dataframe para verificação da etapa anterior

24- Eliminando o backup feito na etapa 22:

Como esperado, o backup não se mostrou relevante, sendo removido nesta etapa.

25- Visualizando o Dataframe para verificação da etapa anterior

26- Dividindo dados da coluna de praças:

Devido a uma particularidade na divisão dos dados no dataset, acabamos precisando fazer a divisão dos valores presentes na coluna 'Praca' para que fosse possível obter alguns dados específicos de forma individual e distribuídos em colunas.

27- Dividindo dados da coluna de praças e salvando em colunas:

Seguindo o racional da etapa 26, agora, os dados já divididos são salvos em novas colunas para melhor manipulação e visualização.

28- Excluindo colunas resultantes da divisão:

Como subproduto da divisão e formatação em coluna dos dados contidos em 'Praca', acabamos por obter diversas novas colunas não pertinentes para o estudo. Portanto, nesse passo usamos o comando 'df.drop' para eliminá-las.

29- Visualizando o Dataframe para verificação da etapa anterior

30- Dividindo dados da coluna 'BRUF' e salvando em colunas:

Ainda não sendo suficiente a primeira divisão (feita no passo 26), recorremos a uma segunda divisão. Porém, dessa vez, utilizando a ' / ' como referência para a separação. Finalmente obtendo o que precisávamos para as correlações pretendidas.

31- Eliminando colunas repetidas

32- Visualizando o Dataframe para verificação da etapa anterior

33- Exportando o Dataframe JSON tratado para CSV na '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 json 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.