Projetos de Análise de Dados com Linguagem Python

Projeto 3 - Técnicas de Limpeza e Tratamento de Valores Ausentes Para Análise de Dados

Pacotes Python Usados no Projeto

In [ ]:

In [ ]:

```
# Imports
import math
import sys, os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

Carregando os Dados

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

In [ ]:

```
# Criamos uma lista para identificar valores ausentes
lista_labels_valores_ausentes = ["n/a", "na", "undefined"]
```

In [ ]:

```
# Carrega o dataset
dataset_dsa = pd.read_csv("dataset.csv", na_values =
lista_labels_valores_ausentes)
```

In [ ]:

```
# Shape
dataset_dsa.shape
```

In [ ]:

```
# Amostra de dados
dataset_dsa.head()
```

In [ ]:
```
# Define o número total de colunas para mostrar ao imprimir o dataframe
pd.set_option('display.max_columns', 100)
```

In [ ]:
```
# Amostra de dados
dataset_dsa.head()
```

In [ ]:
```
# Carregando o dicionário de dados
dicionario = pd.read_excel("dicionario.xlsx")
```

In [ ]:
```
# Shape
dicionario.shape
```

In [ ]:
```
# Amostra de dados
dicionario.head(10)
```

In [ ]:
```
# Define um valor grande para a largura máxima da coluna
pd.set_option('display.max_colwidth', 100)
```

In [ ]:
```
# Amostra de dados
dicionario.head(60)
```

Análise Exploratória

In [ ]:
```
# Info
dataset_dsa.info()
```

In [ ]:
```
# Estatísticas descritivas
dataset_dsa.describe()
```

Não faz sentido calcular estatísticas descritivas para Beared Id, IMSI, MSISDN / Number e IMEI. Mas o método describe() calcula as estatísticas de todas as colunas numéricas. Essas estatísticas estão sendo calculadas antes que os dados sejam limpos. Portanto, pode haver mudanças depois que os valores ausentes e outliers são tratados.

In [ ]:
```
# Shape
dataset_dsa.shape
```

In [ ]:
```
# Shape
dicionario.shape
```

Existem 150.001 linhas e 55 colunas no dataframe. No entanto, temos 56 colunas com seus nomes e descrições no dicionário. Isso significa que há uma coluna descrita, mas não incluída no dataframe. Vamos identificar qual é a coluna faltante.

In [ ]:
```
# Concatena os dataframes
df_compara_colunas = pd.concat([pd.Series(dataset_dsa.columns.tolist()),
dicionario['Fields']],
                               axis = 1)
```

In [ ]:
```
df_compara_colunas.columns
```

In [ ]:
```
# Renomeia as colunas
df_compara_colunas.rename(columns = {0: 'Coluna no Dataset', 'Fields':
'Coluna no Dicionario'},
                          inplace = True)
```

In [ ]:
```
# Visualiza
df_compara_colunas
```

"Dur. (Ms)" é ignorado no dataset como visto no índice 1 em df_compara_colunas. É aqui que a ordem das colunas começou a mudar.

Mas o mesmo nome de coluna "Dur. (Ms)" aparece no dataset no índice 5, enquanto o arquivo de dicionário nos diz que é "Dur. (S)" no índice 6. Como as medidas de ambas as colunas diferem conforme mostrado em seus nomes, nós precisamos verificar qual está certo. Para investigar isso, usaremos a coluna "Dur. (Ms) .1" que se encontra nos índices 28 e 29 no dataset e no arquivo de dicionário, respectivamente.

In [ ]:
```
dataset_dsa[['Dur. (ms)', 'Dur. (ms).1']]
```

Parece que a coluna "Dur. (Ms)" é medida em segundos. Portanto, vamos renomeá-la apropriadamente. Vamos também renomear algumas das colunas para que fiquem claras como sua descrição e sigam o estilo de nomenclatura de outras colunas.

In [ ]:
```
# Renomeia colunas
dataset_dsa.rename(columns = {'Dur. (ms)': 'Dur (s)',
                              'Dur. (ms).1': 'Dur (ms)',
                              'Start ms': 'Start Offset (ms)',
                              'End ms': 'End Offset (ms)'},
                   inplace = True)
```

In [ ]:
```
# Lista de colunas do dataset
dataset_dsa.columns.tolist()
```

In [ ]:

```
dataset_dsa.shape
```

Etapa 1 - Tratamento de Valores Ausentes

- 1- Identificando Valores Ausentes
- 2- Drop de Colunas
- 3- Imputação com Preenchimento Reverso
- 4- Imputação com Preenchimento Progressivo
- 5- Imputação de Variáveis Categóricas
- 6- Drop de Linhas

1.1. Identificando Valores Ausentes

In [ ]:

```
# Função que calcula o percentual de valores ausentes
def func_dsa_calc_percent_valores_ausentes(df):

    # Calcula o total de células no dataset
    totalCells = np.product(df.shape)

    # Conta o número de valores ausentes por coluna
    missingCount = df.isnull().sum()

    # Calcula o total de valores ausentes
    totalMissing = missingCount.sum()

    # Calcula o percentual de valores ausentes
    print("O dataset tem", round(((totalMissing/totalCells) * 100), 2),
"%", "de valores ausentes.")
```

In [ ]:

```
# Verifica o percentual de valores ausentes
func_dsa_calc_percent_valores_ausentes(dataset_dsa)
```

In [ ]:

```
# Função para calcular valores ausentes por coluna
def func_dsa_calc_percent_valores_ausentes_coluna(df):

    # Total de valores ausentes
    mis_val = df.isnull().sum()

    # Porcentagem de valores ausentes
    mis_val_percent = 100 * mis_val / len(df)

    # Tipo de dado das colunas com valores ausentes
```

```
    mis_val_dtype = df.dtypes

    # Cria uma tabela com os resultados
    mis_val_table = pd.concat([mis_val, mis_val_percent, mis_val_dtype],
axis=1)

    # Renomear as colunas
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Valores Ausentes', 1 : '% de Valores Ausentes', 2:
'Dtype'})

    # Classifica a tabela por porcentagem de valores ausentes de forma
decrescente e remove colunas sem valores faltantes
    mis_val_table_ren_columns =
mis_val_table_ren_columns[mis_val_table_ren_columns.iloc[:,0] !=
0].sort_values('% de Valores Ausentes', ascending = False).round(2)

    # Print
    print ("O dataset tem " + str(df.shape[1]) + " colunas.\n"
        "Encontrado: " + str(mis_val_table_ren_columns.shape[0]) + "
colunas que têm valores ausentes.")

    if mis_val_table_ren_columns.shape[0] == 0:
        return

    # Retorna o dataframe com informações ausentes
    return mis_val_table_ren_columns
```

In [ ]:

```
# Cria tabela com valores ausentes
df_missing = func_dsa_calc_percent_valores_ausentes_coluna(dataset_dsa)
```

In [ ]:

```
# Visualiza
df_missing
```

Normalmente, colunas com mais de 50% de valores ausentes devem ser removidas. Entre 30 e 50% é opcional.

Neste projeto vamos remover colunas cujos valores ausentes representem mais de 30% da variável, pois temos um número muito grande de colunas com valores ausentes e, portanto, muito trabalho. Vamos tratar às variáveis com percentual baixo e deletar aquelas que tiverem percentual alto de valores ausentes.

1.2. Drop de Colunas

In [ ]:

```
# Colunas que serão removidas
colunas_para_remover = df_missing[df_missing['% de Valores Ausentes'] >=
30.00].index.tolist()
```

In [ ]:

```
# Colunas que serão removidas
colunas_para_remover
```

Mesmo que as variáveis "TCP" tenham muitos valores ausentes, em vez de removê-las, iremos aplicar imputação a essas variáveis, uma vez que elas podem ser necessárias para nossa análise posterior.

In [ ]:

```
# Colunas que serão removidas
colunas_para_remover = [col for col in colunas_para_remover if col not in
['TCP UL Retrans. Vol (Bytes)',

'TCP DL Retrans. Vol (Bytes)']]
```

In [ ]:

```
# Colunas que serão removidas
colunas_para_remover
```

In [ ]:

```
# Drop das colunas e cria outro dataframe
dataset_dsa_limpo = dataset_dsa.drop(colunas_para_remover, axis = 1)
```

In [ ]:

```
# Shape
dataset_dsa_limpo.shape
```

Agora vamos verificar o status dos valores ausentes no dataframe modificado.

In [ ]:

```
func_dsa_calc_percent_valores_ausentes(dataset_dsa_limpo)
```

In [ ]:

```
func_dsa_calc_percent_valores_ausentes_coluna(dataset_dsa_limpo)
```

1.3. Imputação com Preenchimento Reverso

Uma vez que as porcentagens de valores ausentes de 'TCP UL Retrans. Vol (Bytes) 'e' TCP DL Retrans. Vol (Bytes) 'são muito altos, iremos aplicar imputação nos valores ausentes com o método de preenchimento reverso.

Nesse caso, usar um único valor como média ou mediana não é aconselhável, pois pode alterar nossos dados de uma forma indesejada, tornando a maioria dos valores igual a um único valor.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html

In [ ]:

```
# Imputação de valores ausentes usando backward fill
```

```
# method = 'bfill': Bfill ou backward-fill propaga o primeiro valor não nulo observado para trás até que
# outro valor não nulo seja encontrado.
def func_dsa_fix_missing_bfill(df, col):

    count = df[col].isna().sum()

    df[col] = df[col].fillna(method = 'bfill')

    print(f"{count} valores ausentes na coluna {col} foram substituídos usando o método de preenchimento reverso.")
```

In [ ]:

```
# Imputação com Preenchimento Reverso na variável 'TCP UL Retrans. Vol (Bytes)'
func_dsa_fix_missing_bfill(dataset_dsa_limpo, 'TCP UL Retrans. Vol (Bytes)')
```

In [ ]:

```
# Imputação com Preenchimento Reverso na variável 'TCP DL Retrans. Vol (Bytes)'
func_dsa_fix_missing_bfill(dataset_dsa_limpo, 'TCP DL Retrans. Vol (Bytes)')
```

1.4. Imputação com Preenchimento Progressivo

In [ ]:

```
func_dsa_calc_percent_valores_ausentes_coluna(dataset_dsa_limpo)
```

Avg RTT DL (ms) e Avg RTT UL (ms) têm as próximas porcentagens mais altas de valores ausentes com cerca de 18,5% cada. Vamos verificar se as variáveis estão enviesadas (não seguem uma distribuição normal) usando o método skew(), que retorna o coeficiente de assimetria.

In [ ]:

```
dataset_dsa_limpo['Avg RTT DL (ms)'].skew(skipna = True)
```

In [ ]:

```
dataset_dsa_limpo['Avg RTT UL (ms)'].skew(skipna = True)
```

- Se a assimetria estiver entre -0,5 e 0,5, os dados são bastante simétricos
- Se a assimetria estiver entre -1 e - 0,5 ou entre 0,5 e 1, os dados estão moderadamente inclinados
- Se a assimetria for menor que -1 ou maior que 1, os dados estão altamente enviesados

Visto que ambas as colunas Avg RTT DL (ms) e Avg RTT UL (ms) são fortemente enviesadas positivamente é aconselhável não imputá-las com sua média. Portanto, usaremos o preenchimento progressivo.

In [ ]:

```
# Imputação de valores ausentes usando forward fill (preenchimento progressivo)
# method = 'ffill': Ffill ou forward-fill propaga o último valor não nulo observado para frente até que
# outro valor não nulo seja encontrado
def func_dsa_fix_missing_ffill(df, col):

    count = df[col].isna().sum()

    df[col] = df[col].fillna(method = 'ffill')

    print(f"{count} valores ausentes na coluna {col} foram substituídos usando o método de preenchimento progressivo.")
```

In [ ]:

```
# Imputação com Preenchimento Progressivo
func_dsa_fix_missing_ffill(dataset_dsa_limpo, 'Avg RTT DL (ms)')
```

In [ ]:

```
# Imputação com Preenchimento Progressivo
func_dsa_fix_missing_ffill(dataset_dsa_limpo, 'Avg RTT UL (ms)')
```

Checamos novamente os valores ausentes.

In [ ]:

```
func_dsa_calc_percent_valores_ausentes(dataset_dsa_limpo)
```

In [ ]:

```
func_dsa_calc_percent_valores_ausentes_coluna(dataset_dsa_limpo)
```

1.5. Imputação de Variáveis Categóricas

In [ ]:

```
dataset_dsa_limpo.info()
```

Visto que "Handset Type" e "Handset Manufacturer" são colunas categóricas, é melhor imputá-las com o valor "unknown" para que não enviesemos os dados.

In [ ]:

```
# Preenche valor NA
def func_dsa_fix_missing_value(df, col, value):

    count = df[col].isna().sum()

    df[col] = df[col].fillna(value)

    if type(value) == 'str':
```

```
        print(f"{count} valores ausentes na coluna {col} foram
substituídos por '{value}'.")
    else:
        print(f"{count} valores ausentes na coluna {col} foram
substituídos por {value}.")
```

In [ ]:

```
# Imputação de variáveis categóricas
func_dsa_fix_missing_value(dataset_dsa_limpo, 'Handset Type', 'unknown')
func_dsa_fix_missing_value(dataset_dsa_limpo, 'Handset Manufacturer',
'unknown')
```

Checamos novamente os valores ausentes.

In [ ]:

```
func_dsa_calc_percent_valores_ausentes(dataset_dsa_limpo)
```

In [ ]:

```
func_dsa_calc_percent_valores_ausentes_coluna(dataset_dsa_limpo)
```

1.6. Drop de Linhas

Uma vez que apenas 0.17% do dataset contêm valor ausente e o número total de linhas é de cerca de 150000, descartar essas linhas não terá um impacto negativo perceptível.

In [ ]:

```
# Drop de linhas com valores ausentes
def func_dsa_drop_linhas_com_na(df):

    old = df.shape[0]

    df.dropna(inplace = True)

    new = df.shape[0]

    count = old - new

    print(f"{count} linhas contendo valores ausentes foram descartadas.")
```

In [ ]:

```
# Drop de linhas com valores ausentes
func_dsa_drop_linhas_com_na(dataset_dsa_limpo)
```

In [ ]:

```
func_dsa_calc_percent_valores_ausentes(dataset_dsa_limpo)
```

In [ ]:

```
# Shape
```

```
dataset_dsa_limpo.shape
```

In [ ]:

```
%watermark -a "Data Science Academy"
```

Etapa 2 - Conversão de Tipos de Dados

In [ ]:

```
dataset_dsa_limpo.dtypes
```

In [ ]:

```
dataset_dsa_limpo.select_dtypes(include = 'object').columns.tolist()
```

In [ ]:

```
dataset_dsa_limpo.head()
```

Observando as colunas, podemos perceber que as colunas "Start" e "End" são, na verdade, valores de data e hora, embora sejam rotuladas como objetos pelo pandas. Além dessas duas colunas, todas as outras colunas com tipos de dados de objeto são, na verdade, valores de string. Portanto, vamos converter essas colunas em seus tipos de dados apropriados.

In [ ]:

```
# Função que converte para datetime
def func_dsa_convert_to_datetime(df, columns):
    for col in columns:
        df[col] = pd.to_datetime(df[col])
```

In [ ]:

```
# Converte para datetime
func_dsa_convert_to_datetime(dataset_dsa_limpo, ['Start', 'End'])
```

In [ ]:

```
dataset_dsa_limpo.dtypes
```

In [ ]:

```
# Extrai as colunas do tipo object
string_columns = dataset_dsa_limpo.select_dtypes(include =
'object').columns.tolist()
```

In [ ]:

```
# Visualiza
string_columns
```

In [ ]:

```
# Função que converte para string
def func_dsa_convert_to_string(df, columns):
    for col in columns:
        df[col] = df[col].astype("string")
```

In [ ]:

```
# Converte para string
func_dsa_convert_to_string(dataset_dsa_limpo, string_columns)
```

In [ ]:

```
dataset_dsa_limpo.dtypes
```

Também sabemos que Bearer Id, IMSI, MSISDN / Number, IMEI são números únicos usados para identificação. Portanto, para melhor legibilidade (e facilitar os filtros usados em análises posteriores), vamos alterá-los de float64 para int64.

In [ ]:

```
# Lista de colunas para conversão
int_cols = ['Bearer Id', 'IMSI', 'MSISDN/Number', 'IMEI']
```

In [ ]:

```
# Função que converte para int
def func_dsa_convert_to_int(df, columns):
    for col in columns:
        df[col] = df[col].astype("int64")
```

In [ ]:

```
# Converte para int
func_dsa_convert_to_int(dataset_dsa_limpo, int_cols)
```

In [ ]:

```
dataset_dsa_limpo.dtypes
```

In [ ]:

```
# Função para o drop de linhas duplicadas
def func_dsa_drop_duplicates(df):
    old = df.shape[0]
    df.drop_duplicates(inplace = True)
    new = df.shape[0]
    count = old - new
    if (count == 0):
        print("Nenhuma linha duplicada foi encontrada.")
    else:
        print(f"{count} linhas duplicadas foram encontradas e
removidas.")
```

In [ ]:

```
# Vamos checar se há registros duplicados e, se houver, removemos
func_dsa_drop_duplicates(dataset_dsa_limpo)
```

Como vimos na seção de limpeza da coluna, temos duas colunas de duração, uma em segundos e a outra em microssegundos. Vamos verificar se os valores são iguais convertendo os microssegundos em segundos.

In [ ]:

```
# Função de conversão de unidade de tempo
def func_dsa_converte_unidade(df, columns, factor):
    for col in columns:
        df[col] = df[col] * factor
```

In [ ]:
```
# Retorna as linhas das duas colunas
temp_df = dataset_dsa_limpo[['Dur (s)', 'Dur (ms)']].copy()
```

In [ ]:
```
temp_df.head()
```

In [ ]:
```
# Aplica a função
func_dsa_converte_unidade(temp_df, ['Dur (ms)'], 1/1000)
```

In [ ]:
```
temp_df.head()
```

In [ ]:
```
# Comparação
temp_df['Resultado_Comparacao'] = (temp_df['Dur (s)'] == temp_df['Dur
(ms)'].apply(math.floor))
```

In [ ]:
```
temp_df
```

In [ ]:
```
# As duas colunas são iguais?
print(all(temp_df['Resultado_Comparacao']))
```

Isso prova que, quando arredondadas, essas duas colunas são iguais. Portanto, manteremos "Dur (ms)", pois é mais preciso, e removeremos "Dur (s)".

In [ ]:
```
# Função para o drop de colunas
def func_dsa_drop_columns(df, columns):
    df.drop(columns, axis = 1, inplace = True)
    count = len(columns)
    if count == 1:
        print(f"{count} coluna foi descartada.")
    else:
        print(f"{count} colunas foram descartadas.")
```

In [ ]:
```
# Drop de coluna
func_dsa_drop_columns(dataset_dsa_limpo, ['Dur (s)'])
```

Etapa 3 - Tratamento de Outliers

In [ ]:
```
# Imagem
img_dsa = mpimg.imread('outliers.png')
plt.imshow(img_dsa)
plt.axis('off')
```

```
plt.show()
```

In [ ]:

```
dataset_dsa_limpo.shape
```

In [ ]:

```
import pandas as pd
import numpy as np

# Define a classe TrataOutlier
class TrataOutlier:

    # Construtor da classe que inicializa com um DataFrame
    def __init__(self, df: pd.DataFrame) -> None:
        self.df = df

    # Função para contar outliers nas colunas especificadas
    def count_outliers(self, Q1, Q3, IQR, columns):

        # Define o limite de corte para considerar um valor como outlier
        cut_off = IQR * 1.5

        # Cria um DataFrame temporário com valores booleanos indicando
outliers
        temp_df = (self.df[columns] < (Q1 - cut_off)) | (self.df[columns]
> (Q3 + cut_off))

        # Retorna a contagem de outliers para cada coluna
        return [len(temp_df[temp_df[col] == True]) for col in temp_df]

    # Função para calcular a assimetria das colunas especificadas
    def calc_skew(self, columns=None):

        # Se nenhuma coluna for especificada, utiliza todas as colunas do
DataFrame
        if columns == None:
            columns = self.df.columns

        # Retorna a medida de assimetria para cada coluna
        return [self.df[col].skew() for col in columns]

    # Função para calcular a porcentagem dos valores em relação a 146887
(número de linhas até aqui)
    def percentage(self, list):
```

```
        return [str(round(((value/146887) * 100), 2)) + '%' for value in
list]

    # Função para remover outliers nas colunas especificadas
    def remove_outliers(self, columns):
        for col in columns:

            # Calcula os quantis Q1 e Q3
            Q1, Q3 = self.df[col].quantile(0.25),
self.df[col].quantile(0.75)

            # Calcula a amplitude interquartil (IQR)
            IQR = Q3 - Q1

            # Define os limites para considerar um valor como outlier
            cut_off = IQR * 1.5
            lower, upper = Q1 - cut_off, Q3 + cut_off

            # Remove os valores considerados outliers
            self.df = self.df.drop(self.df[self.df[col] > upper].index)
            self.df = self.df.drop(self.df[self.df[col] < lower].index)

    # Função para substituir outliers pelos valores dos fences nas
colunas especificadas
    def replace_outliers_with_fences(self, columns):
        for col in columns:

            # Calcula os quantis Q1 e Q3
            Q1, Q3 = self.df[col].quantile(0.25),
self.df[col].quantile(0.75)

            # Calcula a amplitude interquartil (IQR)
            IQR = Q3 - Q1

            # Define os limites para considerar um valor como outlier
            cut_off = IQR * 1.5
            lower, upper = Q1 - cut_off, Q3 + cut_off

            # Substitui outliers pelos valores dos fences
            self.df[col] = np.where(self.df[col] > upper, upper,
self.df[col])
```

```
            self.df[col] = np.where(self.df[col] < lower, lower,
self.df[col])

    # Função para obter um resumo estatístico das colunas especificadas
    def getOverview(self, columns) -> None:

        # Calcula diversas estatísticas para as colunas
        min = self.df[columns].min()
        Q1 = self.df[columns].quantile(0.25)
        median = self.df[columns].quantile(0.5)
        Q3 = self.df[columns].quantile(0.75)

        max = self.df[columns].max()
        IQR = Q3 - Q1
        skew = self.calc_skew(columns)
        outliers = self.count_outliers(Q1, Q3, IQR, columns)
        cut_off = IQR * 1.5
        lower, upper = Q1 - cut_off, Q3 + cut_off

        # Define os nomes das colunas para o novo DataFrame
        new_columns = ['Nome de Coluna',
                       'Min',
                       'Q1',
                       'Median',
                       'Q3',
                       'Max',
                       'IQR',
                       'Lower fence',
                       'Upper fence',
                       'Skew',
                       'Num_Outliers',
                       'Percent_Outliers' ]

        # Cria um novo DataFrame com as estatísticas calculadas
        data = zip([column for column in self.df[columns]], min, Q1,
median, Q3, max, IQR, lower, upper, skew, outliers,
self.percentage(outliers))
        new_df = pd.DataFrame(data = data, columns = new_columns)

        # Define 'Nome de Coluna' como o índice do novo DataFrame
        new_df.set_index('Nome de Coluna', inplace = True)
```

```
        # Retorna o novo DataFrame ordenado pelo número de outliers
        return new_df.sort_values('Num_Outliers', ascending = False).transpose()
```

In [ ]:

```
# Cria o objeto trata outlier
trata_outlier = TrataOutlier(dataset_dsa_limpo)
```

In [ ]:

```
# Lista de colunas float64
lista_colunas = dataset_dsa_limpo.select_dtypes('float64').columns.tolist()
```

In [ ]:

```
lista_colunas
```

In [ ]:

```
# Visão geral dos outliers
trata_outlier.getOverview(lista_colunas)
```

In [ ]:

```
# Replace dos outliers
trata_outlier.replace_outliers_with_fences(lista_colunas)
```

In [ ]:

```
# Visão geral dos outliers
trata_outlier.getOverview(lista_colunas)
```

Entregando o Resultado da Análise aos Tomadores de Decisão

Ter a soma dos volumes de dados de upload e download para cada aplicativo como um total pode ser necessário para análises?

In [ ]:

```
dataset_dsa_limpo['Social Media Data Volume (Bytes)'] = dataset_dsa_limpo['Social Media UL (Bytes)'] + dataset_dsa_limpo['Social Media DL (Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Google Data Volume (Bytes)'] = dataset_dsa_limpo['Google UL (Bytes)'] + dataset_dsa_limpo['Google DL (Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Email Data Volume (Bytes)'] = dataset_dsa_limpo['Email UL (Bytes)'] + dataset_dsa_limpo['Email DL (Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Youtube Data Volume (Bytes)'] =
dataset_dsa_limpo['Youtube UL (Bytes)'] + dataset_dsa_limpo['Youtube DL
(Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Netflix Data Volume (Bytes)'] =
dataset_dsa_limpo['Netflix UL (Bytes)'] + dataset_dsa_limpo['Netflix DL
(Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Gaming Data Volume (Bytes)'] =
dataset_dsa_limpo['Gaming UL (Bytes)'] + dataset_dsa_limpo['Gaming DL
(Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Other Data Volume (Bytes)'] = dataset_dsa_limpo['Other
UL (Bytes)'] + dataset_dsa_limpo['Other DL (Bytes)']
```

In [ ]:

```
dataset_dsa_limpo['Total Data Volume (Bytes)'] = dataset_dsa_limpo['Total
UL (Bytes)'] + dataset_dsa_limpo['Total DL (Bytes)']
```

In [ ]:

```
dataset_dsa_limpo.info()
```

In [ ]:

```
dataset_dsa_limpo.shape
```

In [ ]:

```
dataset_dsa_limpo.head()
```

Salvando os Dados Após a Limpeza

In [ ]:

```
# Salvando os dados
dataset_dsa_limpo.to_csv('dataset_limpo.csv')
```

In [ ]:

Fim