Com o intuito de treinar minhas habilidades de preparação de dados, fui até o Kaggle e escolhi uma biblioteca de dados aleatória para começar a treinar. Como gosto muito de música e acho que tenho um pouco mais de conhecimento sobre, escolhi o repositório *History of music (British Library)* mais especificamente o arquivo csv sobre as músicas em si (*records.csv*) que conta com 1.045.508 linhas na tabela. Criei um Jupyter notebook pela comand line do Anaconda e comecei a exploração.

Primeiramente importei todas as ferramentas que poderia usar, o arquivo csv e inspecionei suas primeiras linhas. Ao inspecionar suas colunas já percebi que todos os números de identificação, exceto *BL record ID* e *BL shelfmark*, eram muito inconsistentes, e isso se explica vendo de onde esses dados vieram, pois sabendo que são frutos de uma iniciativa de pesquisa da biblioteca britânica que faz sentido os identificadores relacionados a ela serem consistentes.

Continuando as etapas iniciais ao se explorar dados (vendo os valores únicos de cada tabela, verificando variáveis categóricas, …), ao ver as instâncias nulas de cada coluna percebi como a parte das colunas referente aos números identificadores tinha muito valores nulos:

```
records.isnull().sum()
```

```
BL record ID                          0
Composer                          42059
Composer life dates              583223
Title                                22
Publication date (standardised)   21494
Publication date (not standardised)  15216
Place of publication             329160
Publisher                        389242
ISBN                            1020151
ISMN                            1034661
Publisher number                 901293
BL shelfmark                       1773
dtype: int64
```

A coluna ISBN tendo 1.020.151 entradas nulas, o ISMN tendo 1.034.661 e Publisher number com 901293, mas como esses eram ID 's essas informações não tinha como se conseguir. Passando para as próximas colunas percebi como algumas coisas eu podia relacionar para conseguir mais informações:

- Se o atributo da data de publicação de uma linha for nulo mas a data não padronizada não for, eu posso tentar transformar a informação da data não padronizada em uma data de publicação comum
- Pelo lugar de publicação, pode se conseguir quem publicou por comparar com os que mais publicam naquele lugar nas linhas em que essas informações existem
- Se uma linha tiver o compositor da música mas não o seu período de vida, pode se averiguar se outras instâncias com esse compositor tenham essa informação e então adicionar o seu período de vida onde não tem

Primeiramente comecei pela etapa da data de publicação pois ela é uma variável numérica mais simples de se trabalhar e mais quantificável no processo de avaliação da eficácia do que seja lá o que eu for fazer. Vendo uma descrição da coluna é possível retirar mais algumas conclusões:

```
records['Publication date (standardised)'].describe()
```

```
count   1024014.000000
mean       1914.943543
std          56.749582
min        1018.000000
25%        1885.000000
50%        1920.000000
75%        1957.000000
max        8008.000000
Name: Publication date (standardised), dtype: float64
```

A média de ano de publicação é 1914, as músicas aqui guardadas vão de 1018 até o ano de…8008??? Assim já sabia por onde começar, retirando *outliers* Mas antes estava cansado de digitar esse nome de coluna gigante então mudei ele para 'Publication date'. Agora voltando ao processo anterior:

```
media = int(records['Publication date'].mean())
for index, linha in records.iterrows():
    if(linha['Publication date'] > 2023):
        records.loc[index, 'Publication date'] = media
```

E checando novamente:

```
records['Publication date'].describe()
```

```
count   1024014.000000
mean       1914.937158
std          56.428340
min        1018.000000
25%        1885.000000
50%        1920.000000
75%        1957.000000
max        2019.000000
Name: Publication date, dtype: float64
```

Com isso feito, queria trocar o tipo dessa coluna para um int64 já que não fazia sentido um ano fixo ser um float64, mas ao tentar fazer isso me lembrei dos valores nulos, e como não queria já substituir eles só para mudar seu tipo resolvi trabalhar com os valores nulos primeiro.

```
musicas_sem_data = records[records["Publication date"].isnull()]
len(musicas_sem_data)
```

```
21494
```

Primeiramente, separei as músicas que não possuem uma data de publicação, e como podemos ver na imagem, existem 21494 linhas sem essa informação.

```
musicas_possiveis_arrumar_data = musicas_sem_data[
    musicas_sem_data["Publication date (not standardised)"].notnull()
]
len(musicas_possiveis_arrumar_data)
```

```
18974
```

Dessas 21494 linhas, 18974 continham a data de publicação não padronizada, logo poderiam ter a informação da data de publicação adquirida dessa

coluna. Logo fui ver quais eram as possíveis informações que estariam na coluna de datas não padronizadas:

```
musicas_possiveis_arrumar_data["Publication date (not standardised)"].unique()
```

```
array(['Mas', 'e', 'Mass, printed', 'Ohi', 'at Covent Garden', 'sic',
       '861', 'si', 'Mass', '87', 'b', 'pp', '(Saale)',
       'by Arthur H Mann', 'a', '199', '13', '1', '33', '188', 'u', '178',
       '(-iii)', '2 pt', '189', 'Ont', '194', 'printed', '191', 'Anvie',
       'Ausgewählte Gesänge des Thomanerchores, etc No', '4', 'oru', 'hi',
       'I>Mass', 'I>N', 'Bosto', 'Hambur', 'c', 'Conn',
       "The 'Chrome' Album, etc", '74a', 'rea', '- Le Monde musical',
       ',Mas', 'U S', 'Ms', 'Breme', 'York', '330', '41', 'ongma', 'r',
       'Genev', '(London)', '12', 'London', 'ohn', 'harles an', 'braha',
       'oh', 'amue', 'Samuel', '(Regensburg)', '67', 'N Y', 'New Yor',
       'illia', 'ndre', 'enr', 'N', 'delphi', 'Milano', '1, no',
       'I>Edinbr', 'U S A', 'Vermon', 'Pari', 'eeke', 'Londo',
```

Com isso descobri que haviam mais de 200 valores únicos e que boa parte desses valores não remeteram realmente a anos, mas como no meio vi alguns que poderiam ser convertidos, continuei.

```
guia_datas = { '200-': 2000, '199': 1990,'199-': 1990, 'c199': 1990, '198'
              '198- printing': 1980, '973': 1973, '197': 1970, '197-': 1976
              '196-': 1960, '952': 1952, '195-': 1950, '194': 1940, 'c194':
              '(193-)': 1930, '192-': 1920, '(192-)': 1920, '19-1': 1911, '
              '189': 1890, '889': 1889, '188': 1880, '188-': 1880, '(188-)'
              '(186-)': 1860, '184-': 1840, '(182-)': 1820, '182': 1820, '1
verificacao_guia_datas = np.array(list(guia_datas.keys()))

print(guia_datas['198- printing'])
verificacao_guia_datas
```

```
1980

array(['200-', '199', '199-', 'c199', '198', '198-', '(198)', '(198-)',
       '198- printing', '973', '197', '197-', '(197-)', 'c197', 'c196X',
       '196-', '952', '195-', '194', 'c194', '(194-)', '193-', '(193)',
       '(193-)', '192-', '(192-)', '19-1', '191', 'c191', '191-',
       '(190-)', '189', '889', '188', '188-', '(188-)', '187-', '861',
       '186', '(186-)', '184-', '(182-)', '182', '178', '417', '330',
       '244'], dtype='<U13')
```

Analisando todas as possibilidades que poderiam ser anos, criei um dicionário, guia_datas, para que quando passasse um valor das strings possíveis da coluna de data não padronizada, caso se remetesse à um ano, ele retornasse o ano correspondente àquela string. A variável vareificacao_guia_datas apenas servirá

como validação para o processo das atualizações das datas, como demonstrado a seguir:

```
for index, linha_musica in musicas_possiveis_arrumar_data.iterrows():
    dica_data = linha_musica["Publication date (not standardised)"]

    if(dica_data in verificacao_guia_datas):
        records.loc[index, 'Publication date'] = guia_datas[dica_data]
```

Fazendo um looping pelas linhas do DataFrame musicas_possiveis_arrumar_data, eu separo a informação da data em dica_data e vejo se aquela dica_data é possível transformar em um ano, ou seja, verifico se ela é um valor que existe nas strings que haviam sido separadas em verificacao_guia_datas. Caso esteja na lista, o valor então é substituído no DataFrame original, records, sendo colocado o valor de retorno de guia_datas, quando se passa a dica_data para ela.

Rodando novamente as células das informações sobre as músicas com valores nulos na coluna de data de publicação podemos ver…

```
musicas_sem_data = records[records["Publication date"].isnull()]
len(musicas_sem_data)
```

```
21362
```

```
musicas_possiveis_arrumar_data = musicas_sem_data[
    musicas_sem_data["Publication date (not standardised)"].notnull()
]
len(musicas_possiveis_arrumar_data)
```

```
18842
```

Que comparado ao resultado anterior, somente foram mudadas 132 linhas, que correspondem a apenas aproximadamente 0.01% do total das entradas, logo mudando…

```
records['Publication date'].describe()
```

```
count   1024146.000000
mean       1914.939308
std          56.437256
min        1018.000000
25%        1885.000000
50%        1920.000000
75%        1957.000000
max        2019.000000
Name: Publication date, dtype: float64
```

Nada do contexto geral dessa coluna. E com isso aprendi a validar minhas ações antes de executá-las, pois se tivesse visto que apenas 132 músicas das 18974 que “poderiam ser arrumadas” continham realmente uma informação de ano, não teria gastado tempo nesse processo, e se tivesse percebido que mesmo que eu mudasse 18974 linhas isso não representaria nem 2% das linhas totais do Dataframe, nem havia começado. Mas pelo menos com isso consegui uma nova informação, a coluna de data de publicação não padronizada não continha somente valores de ano, mas também de lugar. Então uma nova tarefa surgiu após a primeira etapa estipulada. Atualizando tarefas:

1. ~~Se o atributo da data de publicação de uma linha for nulo mas a data não padronizada não for, eu posso tentar transformar a informação da data não padronizada em uma data de publicação comum;~~
2. Se o valor de lugar de publicação for nulo mas houver a informação de data não padronizada, há como conseguir a informação do lugar de lá;
3. Pelo lugar de publicação, pode se conseguir quem publicou por comparar com os que mais publicam naquele lugar nas linhas em que essas informações existem;
4. Se uma linha tiver o compositor da música mas não o seu período de vida, pode se averiguar se outras instâncias com esse compositor tenham essa informação e então adicionar o seu período de vida onde não tem

Agora com minha lição aprendida, primeiro separei as linhas que se encaixam na descrição dessa atividade e vi se o número delas faria uma diferença no Dataframe caso sejam mudadas.

```
musicas_sem_localizacao = records[records['Place of publication'].isnull()]
len(musicas_sem_localizacao)
```

```
329160
```

```
musicas_possiveis_arrumar_localizacao = musicas_sem_localizacao[
    musicas_sem_localizacao['Publication date (not standardised)'].notnull()
]
len(musicas_possiveis_arrumar_localizacao)
```

```
315573
```

Com a verificação da quantidade, 315573 foi considerado um número qualificado para se trabalhar. Agora indo para a validação do impacto que as mudanças fariam no Dataframe. Peguei os valores únicos de data de publicação não padronizada dentro de musicas_possiveis_arrumar_localizacao:

```
valores_unicos = musicas_possiveis_arrumar_localizacao['Publication date (not standardised)'].unique()
len(valores_unicos)
```

```
5004
```

Contendo 5004 entradas diferentes resolvi filtrá-las um pouco pois possivelmente haviam informações inúteis ali, mas como queria ser cauteloso em quais iria retirar comecei por partes. Primeiramente queria retirar os valores numéricos que estavam lá, e após isso ainda sobraram 4634 valores únicos.

```
valores_possiveis =[]
for valor_unico in valores_unicos:
    if(not valor_unico.isnumeric() ):
        valores_possiveis.append(valor_unico)

len(valores_possiveis)
```

```
4634
```

Verificando se isso ainda continuaria válido, é notado que ainda existem 287974 linhas válidas com as colunas já filtradas.

```
len(musicas_possiveis_arrumar_localizacao[
    musicas_possiveis_arrumar_localizacao['Publication date (not standardised)'].isin(valores_possiveis)
])
```

```
287974
```

Anteriormente inspecionando a tabela vi que a menor string que indicava localidade registrada ali era ‘London’ então filtrei pelo tamanho da string também, resultando ainda em 4396 valores únicos.

```
valores_possiveis =[]
for valor_unico in valores_unicos:
    if(not valor_unico.isnumeric() and len(valor_unico) > 5):
        valores_possiveis.append(valor_unico)

len(valores_possiveis)
```

```
4396
```

E para validar as linhas ainda dentro desse grupo verifiquei novamente a célula anterior:

```
len(musicas_possiveis_arrumar_localizacao[
    musicas_possiveis_arrumar_localizacao['Publication date (not standardised)'].isin(valores_possiveis)
])
```

```
10784
```

Que agora só tinha 10784 valores correspondentes, ou seja beirando o 1% das entradas totais do Data Frame, e assim inutilizando o resultado desse processo. Mesmo que essas mudanças fizessem alguma diferença caso esses dados fossem usados para alguma coisa, estou apenas realizando esses processos para refinar os dados utilizados, e assim refinar minhas habilidades de preparação de dados, e eu julguei com base no baixo impacto dessa ação que ela não irá influenciar em nada o contexto geral de minhas atividades. E assim mais um processo “concluído”.

1. ~~Se o atributo da data de publicação de uma linha for nulo mas a data não padronizada não for, eu posso tentar transformar a informação da data não padronizada em uma data de publicação comum;~~
2. ~~Se o valor de lugar de publicação for nulo mas houver a informação de data não padronizada, há como conseguir a informação do lugar de lá;~~
3. Pelo lugar de publicação, pode se conseguir quem publicou por comparar com os que mais publicam naquele lugar nas linhas em que essas informações existem;

4. Se uma linha tiver o compositor da música mas não o seu período de vida, pode se averiguar se outras instâncias com esse compositor tenham essa informação e então adicionar o seu período de vida onde não tem

Começando o processo de conseguir os compositores pelo lugar de publicação, como sempre, primeiro veria se teria algum impacto no Dataframe inteiro caso eu fizesse aquela ação

```
musicas_sem_compositor = records[records["Composer"].isnull()]
len(musicas_sem_compositor)
```

```
42059
```

```
musicas_possiveis_arrumar_compositor = musicas_sem_compositor[
    musicas_sem_compositor["Place of publication"].notnull()
]
len(musicas_possiveis_arrumar_compositor)
```

```
29397
```

Até o momento chegando a 3% das entradas totais, normalmente não consideraria isso uma influência muito grande mas como foi a maior porcentagem de relevância até agora continuei o processo. E para isso, eu sabia que antes de eu começar a substituir as informações direto no Dataframe eu iria precisar das informações já separadas de qual é o compositor mais comuns para cada localidade:

```
valores_unicos = records[records['Place of publication'].notna()]['Place of publication'].unique()
compositores_frequentes_lugar = []
valores_unicos_validos = []

for place in valores_unicos:
    linhas_correspondentes = records[records["Place of publication"] == place]
    moda = linhas_correspondentes["Composer"].mode()
    if(len(moda) > 0):
        compositores_frequentes_lugar.append(moda[0])
        valores_unicos_validos.append(place)
```

A variável valores_unicos guarda os valores únicos de lugar de publicação excluindo os valores nulos e que também registram “NaN”. compositores_frequentes_lugar, que terá a informação do nome dos compositores

mais comuns de cada lugar , e valores_unicos_validos, que terá os valores de lugar de publicação, serão preenchidos conforme o método abaixo, que para cada lugar pego anteriormente em valores_unicos ele pega as linhas que tem o mesmo valor de lugar de publicação e então verifica a moda dos compositores ao decorrer das linhas que correspondem aquele lugar, caso exista algum compositor como moda ela adiciona ele a compositores_frequentes_lugar e armazena o lugar em si na variável valores_unicos_validos.

A razão para a criação dessas duas listas de lugares e compositores foi para depois construir um Dataframe que ajudaria a relacionar os dois, pois sabendo o lugar de publicação, ao lado dele, na mesma linha do Dataframe, conseguiria facilmente a informação de qual é o compositor comum naquele lugar

```
data={'Place of publication': valores_unicos_validos,'Mode composer': compositores_frequentes_lugar}

compositores_moda_lugar = pd.DataFrame(data)
```

Com isso pronto, precisava somente rodar esse bloco de código:

```
for index, musica_arrumar in musicas_possiveis_arrumar_compositor.iterrows():
    lugar_arrumar_compositor = musica_arrumar["Place of publication"]
    compositor_lugar = compositores_moda_lugar[
        compositores_moda_lugar["Place of publication"] == lugar_arrumar_compositor
    ]["Mode composer"]

    records.at[index, "Composer"] = compositor_lugar
```

Mas antes de mudar diretamente essas informações, percebi algumas coisas que não considerei:

- O que acontecerá se um lugar tiver mais de um compositor como sua moda?
- Se houver uma instância que tenha um lugar em que apenas é citada nela mesma, mas que falte a informação sobre o seu compositor, ainda assim teria como preencher essa informação?
- Mesmo que eu coloque o compositor mais comum daquela localidade, há como eu garantir que aquela música pertence a ele?

E isso me fez reconsiderar escolhas passadas:

- Como eu posso garantir que se na coluna de data de publicação não padronizada está escrito “c199” a data referida seria 1990?
- Com que certeza eu pude colocar o último algarismo dos anos faltando ele como 0 ? Ou que “19-1” se referia a 1911?
- "London" era mesmo o lugar com o menor nome possível nesse Dataframe?

Mesmo ainda faltando testar uma de minhas ideias, tendo essas incertezas em mente, decidi acabar meus estudos com esse Dataframe. Creio que os dados coletados tenham uma precariedade de informações dado a quantidade de valores nulos de algumas colunas, mas também confesso que minhas habilidades atuais ainda não estão no escopo de conseguir “limpar” esses dados. Mas independente dos resultados, esses pequenos exercícios me fizeram aprender a pensar mais criticamente em minhas escolhas e em como lidar, de forma técnica, com os dados, quem sabe no futuro eu tente de novo.