

# Documentação do Projeto Final - Equipe 01

## Tema: Saúde

### Contexto

O Projeto Final do Curso de Engenharia de Dados da SoulCode Academy consiste no ETL(Extract, Transform and Load) de 2 Datasets escolhidos pela equipe de acordo com o tema. Onde os dois arquivos devem ser tratados, modelados organizados com a utilização das tecnologias vistas ao longo do curso, Google Cloud Platform, Python, Pandas, PySpark, SparkSQL, Cloud Storage, Big Query e Data Studio. Além do processo de ETL foi feita a análise dos Dados para gerar insights a partir deles.

### Datasets

Os Datasets foram escolhidos a partir do site <https://www.data.gov/>, um site que armazenar dados abertos do Governo dos Estados Unidos. Os datasets selecionados foram os dados de Hospitalização por COVID-19 dos Condados de Connecticut em JSON e os dados de Vacinação dos Condados de Connecticut em CSV.

### Documentação do processo de ETL do Arquivo em Json

Instalando a biblioteca Pandera para validação de Dados.

```
# Instalando Biblioteca Pandera
!pip install pandera
```

Importando as bibliotecas

```
import pandas as pd
import pandera as pa
import json
from google.cloud import storage
```

Utilizando a chave de serviço para acessar o GCP

```
storage_client = storage.Client.from_service_account_json('/content/drive/MyDrive/Dados/Projeto_Final/chave_projeto-fina
```

Acessando o Bucket do Projeto

```
bucket_name = 'projeto-fina
```

Usando For para lista os arquivos presentes no Bucket

```
filename = list(bucket.list_blobs(prefix=''))
for name in filename:
    print(name.name)
```

Carregando o dataset em Json na variável data

```
blob = bucket.blob('dados-brutos/COVID_US_HOSPITALIZATION.json')
data = json.loads(blob.download_as_string(client=None))
```

Criando DataFrame ordenado por Dados utilizando pandas

```
df = pd.DataFrame(data["data"])
df
```

Analizando a base de Dados

```
print(df.info())
```

Visualização do Schema

```
pd.DataFrame(data['meta']['view']['columns'])
```

Deletando colunas que não fazem sentido para a análise de dados

```
df.drop([0,1,2,3,4,5,6,7], axis = 1, inplace=True)
```

Traduzindo o Schema para Português-BR

```
df.rename(columns={
    8:'Data_Atualizacao',
    9:'Condado_Codigo',
    10:'Condado',
    11:'Casos_Total',
    12:'Casos_Confirmados',
    13:'Casos_Provaveis',
    14:'Casos_Total_Taxa',
    15:'Casos_Hospitalizados',
    16:'Obitos_Total',
    17:'Obitos_Confirmados',
    18:'Obitos_Provaveis'
}, inplace=True)
```

Verificando os valores nulos

```
df.isna().sum()
```

Substituindo valores nulos por 0

```
df.replace([None], '0', inplace=True)
```

Modificando o tipo de dado de algumas colunas para fazer a validação

```
df['Data_Atualizacao'] = pd.to_datetime(df['Data_Atualizacao'])
df[['Condado']] = df[['Condado']].astype(str)
```

### Listando Dados Únicos na Coluna 'Condado'

```
sorted(pd.unique(df['Condado']))
```

### Procurando por dado em uma coluna e alterando/acrescentando 'County' aos nomes dos Condados

```
df.loc[df.Condado == 'Fairfield', ['Condado']] = 'Fairfield County'
df.loc[df.Condado == 'Hartford', ['Condado']] = 'Hartford County'
df.loc[df.Condado == 'Litchfield', ['Condado']] = 'Litchfield County'
df.loc[df.Condado == 'Middlesex', ['Condado']] = 'Middlesex County'
df.loc[df.Condado == 'New Haven', ['Condado']] = 'New Haven County'
df.loc[df.Condado == 'New London', ['Condado']] = 'New London County'
df.loc[df.Condado == 'Tolland', ['Condado']] = 'Tolland County'
df.loc[df.Condado == 'Windham', ['Condado']] = 'Windham County'
```

### Alterando o tipo de dados das colunas

```
df[['Condado_Codigo']] = df[['Condado_Codigo']].astype(int)
df[['Casos_Confirmados']] = df[['Casos_Confirmados']].astype(int)
df[['Casos_Provaveis']] = df[['Casos_Provaveis']].astype(int)
df[['Casos_Total_Taxa']] = df[['Casos_Total_Taxa']].astype(float)
df[['Casos_Hospitalizados']] = df[['Casos_Hospitalizados']].astype(int)
df[['Obitos_Total']] = df[['Obitos_Total']].astype(int)
df[['Obitos_Confirmados']] = df[['Obitos_Confirmados']].astype(int)
df[['Obitos_Provaveis']] = df[['Obitos_Provaveis']].astype(int)
```

### Validando os dados do DataFrame usando a biblioteca pandera

```
schema = pa.DataFrameSchema(
    columns = {
        "Data_Atualizacao":pa.Column(pa.DateTime),
        "Condado_Codigo":pa.Column(pa.Int),
        "Condado":pa.Column(pa.String),
        "Casos_Total":pa.Column(pa.Int),
        "Casos_Confirmados":pa.Column(pa.Int),
        "Casos_Provaveis":pa.Column(pa.Int),
        "Casos_Total_Taxa":pa.Column(pa.Float),
        "Casos_Hospitalizados":pa.Column(pa.Int),
        "Obitos_Total":pa.Column(pa.Int),
        "Obitos_Confirmados":pa.Column(pa.Int),
        "Obitos_Provaveis":pa.Column(pa.Int)
    }
)
schema.validate(df)
```

### Conferindo Dados do Dataframe

```
print(df.info())
```

### Fragmentando o DataFrame por Condado e salvando no Bucket do GCP

```
# Criando DataFrame para o Condado Fairfield County
df_Fairfield_County = df.loc[df.Condado == 'Fairfield County']
df_Fairfield_County
# Salvando DataFrame do Condado Fairfield County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finaiscc01')
bucket.blob('dados-tratados/Fairfield_County_Hospitalizacoes.csv').upload_from_string(df_Fairfield_County.to_csv(index=False), 'Fairfield_Co
# Criando DataFrame para o Condado Hartford County
```

```

df_Hartford_County = df.loc[df.Condado == 'Hartford County']
df_Hartford_County
# Salvando DataFrame do Condado Hartford County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Hartford_County_Hospitalizacoes.csv').upload_from_string(df_Hartford_County.to_csv(index=False), 'Hartford_County.csv')

# Criando DataFrame para o Condado Litchfield County
df_Litchfield_County = df.loc[df.Condado == 'Litchfield County']
df_Litchfield_County
# Salvando DataFrame do Condado Litchfield County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Litchfield_County_Hospitalizacoes.csv').upload_from_string(df_Litchfield_County.to_csv(index=False), 'Litchfield_County.csv')

# Criando DataFrame para o Condado Middlesex County
df_Middlesex_County = df.loc[df.Condado == 'Middlesex County']
df_Middlesex_County
# Salvando DataFrame do Condado Middlesex County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Middlesex_County_Hospitalizacoes.csv').upload_from_string(df_Middlesex_County.to_csv(index=False), 'Middlesex_County.csv')

# Criando DataFrame para o Condado New Haven County
df_New_Haven_County = df.loc[df.Condado == 'New Haven County']
df_New_Haven_County
# Salvando DataFrame do Condado New Haven County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/New_Haven_County_Hospitalizacoes.csv').upload_from_string(df_New_Haven_County.to_csv(index=False), 'New_Haven_County.csv')

# Criando DataFrame para o Condado New London County
df_New_London_County = df.loc[df.Condado == 'New London County']
df_New_London_County
# Salvando DataFrame do Condado New London County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/New_London_County_Hospitalizacoes.csv').upload_from_string(df_New_London_County.to_csv(index=False), 'New_London_County.csv')

# Criando DataFrame para o Condado Tolland County
df_Tolland_County = df.loc[df.Condado == 'Tolland County']
df_Tolland_County
# Salvando DataFrame do Condado Tolland County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Tolland_County_Hospitalizacoes.csv').upload_from_string(df_Tolland_County.to_csv(index=False), 'Tolland_County_Hospitalizacoes.csv')

# Criando DataFrame para o Condado Windham County
df_Windham_County = df.loc[df.Condado == 'Windham County']
df_Windham_County
# Salvando DataFrame do Condado Windham County no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Windham_County_Hospitalizacoes.csv').upload_from_string(df_Windham_County.to_csv(index=False), 'Windham_County_Hospitalizacoes.csv')

# Salvando DataFrame normalizado no Bucket do GCP
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Casos_Hospitalizacoes_Obitos_Cidade.csv').upload_from_string(df.to_csv(index=False), 'Casos_Hospitalizacoes_Obitos_Cidade.csv')

```

## Tratamento do Arquivo CSV

Instalando a Biblioteca Pandera (no Colab toda vez em que se inicia o notebook tem que fazer a instalação da biblioteca para ser utilizada)

```
!pip install pandera
```

Importando Bibliotecas para a utilização do pandas e para acessar o bucket do GCP

```

from google.cloud import storage
import pandas as pd
import pandera as pa
import io
from io import BytesIO

```

Criando o client object com a chave Json de acesso ao projeto na GCP

```
storage_client = storage.Client.from_service_account_json("/content/chave_projeto-finalesc01-b94741ddd471.json")
```

### Acessando o Bucket do projeto

```
BUCKET_NAME = 'projeto-finalesc01'

bucket = storage_client.get_bucket(BUCKET_NAME)
```

### Listando os arquivos contidos no Bucket

```
filename = list(bucket.list_blobs(prefix=''))
for name in filename:
    print(name.name)
```

### Carregando o dataset .csv

```
blop = bucket.blob("dados-brutos/COVID-19_Vaccinations_in_the_United_States_County.csv")
data = blop.download_as_string()
```

### Criando um DataFrame utilizando pandas

```
df = pd.read_csv(io.BytesIO(data), encoding='utf-8', sep=',', parse_dates = ["Date"], dayfirst = True)

df
```

### Analizando a base de dados

```
print(df.info())
```

### Traduzindo o Schema para Português-BR

```
df.rename(columns={'Date': 'Data',
                  'FIPS': 'CEPPIF', #Código Estadual Padrão de Processamento de Informações Federais
                  'MMWR_week': 'Relatorio_semanal_morbidade_mortalidade', #controle e prevenção de doenças
                  'Recip_County': 'Condado', #região em que reside o paciente
                  'Recip_State': 'Estado', #estado em que reside o paciente
                  'Series_Complete_Pop_Pct': 'Porcentagem_populacao_vacinada', #Porcentagem de pessoas que estão totalmente vacinadas (têm
                  'Series_Complete_Yes': 'Numero_total_vacinados', #Número total de pessoas que estão totalmente vacinadas (têm segunda dose)
                  'Series_Complete_12Plus': 'Numero_total_12+', #Número total de pessoas com mais de 12 anos que estão totalmente vacinadas
                  'Series_Complete_12PlusPop_Pct': 'Porcentagem_total_12+', #Porcentagem de pessoas com mais de 12 anos que estão totalmente vacinadas
                  'Series_Complete_18Plus': 'Numero_total_18+', #Número total de pessoas com mais de 18 anos que estão totalmente vacinadas
                  'Series_Complete_18PlusPop_Pct': 'Porcentagem_total_18+', #Porcentagem de pessoas com mais de 18 anos que estão totalmente vacinadas
                  'Series_Complete_65Plus': 'Numero_total_65+', #Número total de pessoas com mais de 65 anos que estão totalmente vacinadas
                  'Series_Complete_65PlusPop_Pct': 'Porcentagem_total_65+', #Porcentagem de pessoas com mais de 65 anos que estão totalmente vacinadas
                  'Completeness_pct': 'Porcentagem_total_pessoas_vacinadas', #Representa a proporção de pessoas totalmente vacinadas cujo
                  'Administered_Dose1_Recip': 'Primeira_dose', #Pessoas com pelo menos uma dose por estado de residência
                  'Administered_Dose1_Pop_Pct': 'Porcentagem_primeira_dose', #Porcentagem da População total com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_12Plus': 'Primeira_dose_12+', #Pessoas com de 12 anos com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_12PlusPop_Pct': 'Porcentagem_primeira_dose_12+', #Porcentagem de 12+ Pop com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_18Plus': 'Primeira_dose_18+', #Pessoas com de 18 anos com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_18PlusPop_Pct': 'Porcentagem_primeira_dose_18+', #Porcentagem de Pop 18+ com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_65Plus': 'Primeira_dose_65+', #Pessoas com mais de 65 anos com pelo menos uma dose por estado
                  'Administered_Dose1_Recip_65PlusPop_Pct': 'Porcentagem_primeira_dose_65+', #Porcentagem de Pop com mais de 65 anos com pelo
                  'SVI_CTRY': 'IVS_Categoria', #Índice de Vulnerabilidade Social por Categoria
                  'Series_Complete_Pop_Pct_SVI': 'Porcentagem_total_pop_vacinada', #Porcentagem da população totalmente vacinada / IVS
                  'Series_Complete_12PlusPop_Pct_SVI': 'Porcentagem_total_12+_IVS', #Porcentagem da população com 12+ totalmente vacinadas
                  'Series_Complete_18PlusPop_Pct_SVI': 'Porcentagem_total_18+_IVS', #Porcentagem da população com 18+ totalmente vacinadas
                  'Series_Complete_65PlusPop_Pct_SVI': 'Porcentagem_total_65+_IVS', #Porcentagem da população com 65+ totalmente vacinadas
                  'Metro_status': 'Status_metropolitano', #Status da área metropolitana (se é uma área metropolitana ou não metropolitana)
                  'Series_Complete_Pop_Pct.UR_Equity': 'Porcentagem_total_metropolitana/NaoMetropolitana', #Porcentagem da população total
                  'Series_Complete_12PlusPop_Pct.UR_Equity': 'Porcentagem_total_12+_metropolitana/NaoMetropolitana', #Porcentagem da população com 12+
                  'Series_Complete_18PlusPop_Pct.UR_Equity': 'Porcentagem_total_18+_metropolitana/NaoMetropolitana', #Porcentagem da população com 18+
                  'Series_Complete_65PlusPop_Pct.UR_Equity': 'Porcentagem_total_65+_metropolitana/NaoMetropolitana' #Porcentagem da população com 65+}
```

```
    }, inplace = True)
```

Verificando se a coluna **Relatorio\_semanal\_morbidade\_mortalidade** contém valores únicos

```
#Verificação realizada para saber o que cada código significava
pd.unique(df['Relatorio_semanal_morbidade_mortalidade'])
```

Deletando colunas que não fazem sentido para a análise de dados

```
df.drop(['CEPPIF', 'Relatorio_semanal_morbidade_mortalidade', 'Porcentagem_populacao_vacinada'], axis = 1, inplace = True)
```

1. A coluna **CEPPIF** representa o código da região federal, como já temos o condado e o estado, não será necessária na base de dados.
2. Como não foi encontrado o que representa cada código da coluna **Relatorio\_semanal\_morbidade\_mortalidade**, resolvemos dropar para não atrapalhar a análise de dados.
3. A coluna **Porcentagem\_populacao\_vacinada** faz referência a coluna **Relatorio\_semanal\_morbidade\_mortalidade**, então não faz sentido deixar se não sabemos o que representa cada código da coluna de relatório.

Verificando os valores nulos

```
df.isna().sum()
```

Deletando linhas com dados faltantes

```
#Verificando os valores conditos na coluna 'Status_metropolitano'
pd.unique(df['Status_metropolitano'])
#Deletando valores nulos da coluna 'Status_metropolitano'
df.dropna(subset = ['Status_metropolitano'], inplace = True)
```

```
#Traduzindo os valores conditos na coluna 'Status_metropolitano'

df.loc[df.Status_metropolitano == 'Metro', ['Status_metropolitano']] = 'Metropolitano'
df.loc[df.Status_metropolitano == 'Non-metro', ['Status_metropolitano']] = 'Não Metropolitano'

df.head()
```

```
#Verificando os valores conditos na coluna 'IVS_Categoria'
pd.unique(df['IVS_Categoria'])
```

```
#Deletando valores nulos da coluna 'IVS_Categoria'
df.dropna(subset = ['IVS_Categoria'], inplace = True)
```

```
#Conferindo se os valores nulos das colunas 'Status_metropolitano' e 'IVS_Categoria' foram deletados
df.isna().sum()
```

Substituindo os valores nulos das colunas restantes por 0 para facilitar a análise estatística do DataFrame

```
df.fillna(0, inplace = True)
```

Verificando se os valores nulos foram substituídos

```
df.isna().sum()
```

Modificando o tipo de dado de algumas colunas para fazer a validação

```
#Modificando a coluna 'Numero_total_12+' de float para int
df[['Numero_total_12+']] = df[['Numero_total_12+']].astype(int)

#Modificando a coluna 'Primeira_dose' de float para int
df[['Primeira_dose']] = df[['Primeira_dose']].astype(int)

#Modificando a coluna 'Primeira_dose_12+' de float para int
df[['Primeira_dose_12+']] = df[['Primeira_dose_12+']].astype(int)

#Modificando a coluna 'Primeira_dose_18+' de float para int
df[['Primeira_dose_18+']] = df[['Primeira_dose_18+']].astype(int)

#Modificando a coluna 'Primeira_dose_65+' de float para int
df[['Primeira_dose_65+']] = df[['Primeira_dose_65+']].astype(int)
```

Validando os dados do DataFrame usando a biblioteca panderas

```
schema = pa.DataFrameSchema(
    columns = {
        "Data":pa.Column(pa.DateTime),
        "Condado":pa.Column(pa.String),
        "Estado":pa.Column(pa.String),
        "Numero_total_vacinados":pa.Column(pa.Int),
        "Numero_total_12+":pa.Column(pa.Int),
        "Porcentagem_total_12+":pa.Column(pa.Float),
        "Numero_total_18+":pa.Column(pa.Int),
        "Porcentagem_total_18+":pa.Column(pa.Float),
        "Numero_total_65+":pa.Column(pa.Int),
        "Porcentagem_total_65+":pa.Column(pa.Float),
        "Porcentagem_total_pessoas_vacinadas":pa.Column(pa.Float),
        "Primeira_dose":pa.Column(pa.Int),
        "Porcentagem_primeira_dose":pa.Column(pa.Float),
        "Primeira_dose_12+":pa.Column(pa.Int),
        "Porcentagem_primeira_dose_12+":pa.Column(pa.Float),
        "Primeira_dose_18+":pa.Column(pa.Int),
        "Porcentagem_primeira_dose_18+":pa.Column(pa.Float),
        "Primeira_dose_65+":pa.Column(pa.Int),
        "Porcentagem_primeira_dose_65+":pa.Column(pa.Float),
        "IVS_Categoria":pa.Column(pa.String),
        "Porcentagem_total_pop_vacinada":pa.Column(pa.Float),
        "Porcentagem_total_12+_IVS":pa.Column(pa.Float),
        "Porcentagem_total_18+_IVS":pa.Column(pa.Float),
        "Porcentagem_total_65+_IVS":pa.Column(pa.Float),
        "Status_metropolitano":pa.Column(pa.String),
        "Porcentagem_total_metropolitana/NaoMetropolitana":pa.Column(pa.Float),
        "Porcentagem_total_12+_metropolitana/NaoMetropolitana":pa.Column(pa.Float),
        "Porcentagem_total_18+_metropolitana/NaoMetropolitana":pa.Column(pa.Float),
        "Porcentagem_total_65+_metropolitana/NaoMetropolitana":pa.Column(pa.Float)
    }
)
schema.validate(df)
```

Salvando as transformações realizadas no DataFrame para o Bucket do GCP

```
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados-tratados/Vacinacoes_nos_EUA.csv').upload_from_string(df.to_csv(index=False), 'Vacinacoes_nos_EUA.csv' )
```

## Fragmentando por Condado o DataFrame de Vacinação

Importando Bibliotecas para a utilização do pandas e para acessar o bucket do GCP

```
from google.cloud import storage
import pandas as pd
import io
from io import BytesIO
```

Criando o client object

```
storage_client = storage.Client.from_service_account_json("/content/projeto-finalesc01-b94741ddd471.json")
```

Acessando o Bucket do Projeto

```
BUCKET_NAME = 'projeto-finalesc01'

bucket = storage_client.get_bucket(BUCKET_NAME)
```

Listando os arquivos contidos no Bucke

```
filename = list(bucket.list_blobs(prefix=''))
for name in filename:
    print(name.name)
```

Carregando o dataset .csv

```
blop = bucket.blob("dados-tratados/vacina_8_condados_us.csv")
data = blop.download_as_string()
```

Criando um DataFrame utilizando pandas

```
df = pd.read_csv(io.BytesIO(data), encoding='utf-8', sep=',') #, parse_dates = ["Date"], dayfirst = True
df
```

Deletando colunas que não serão utilizadas na análise de dados

```
df.drop(['Numero_total_12+', 'Porcentagem_total_12+', 'Numero_total_18+', 'Porcentagem_total_18+', 'Porcentagem_total_pessoas_vacinadas', '']
df.head()
```

Criando e renomeando o index

```
df.reset_index(inplace=True)
df = df.rename(columns = {'index':'indice'})
```

Criando dataset's com os 3 condados com mais casos

```
New_Haven_County = df.loc[df.Condado == 'New Haven County']
New_Haven_County = New_Haven_County.loc[341:0:-1]

bucket = storage_client.get_bucket('projeto-finalesc01')
bucket.blob('dados-tratados/New_Haven_County_vacinacao.csv').upload_from_string(New_Haven_County.to_csv(index=False), 'New_Haven_County_vaci
```

```
Fairfield_County = df.loc[df.Condado == 'Fairfield County']
Fairfield_County = Fairfield_County.loc[341:0:-1]

bucket = storage_client.get_bucket('projeto-finalesc01')
bucket.blob('dados-tratados/Fairfield_County_vacinacao.csv').upload_from_string(Fairfield_County.to_csv(index=False), 'Fairfield_County_vaci
```

```
Hartford_County = df.loc[df.Condado == 'Hartford County']
Hartford_County = Hartford_County.loc[341:0:-1]

bucket = storage_client.get_bucket('projeto-finalesc01')
bucket.blob('dados-tratados/Hartford_County_vacinacao.csv').upload_from_string(Hartford_County.to_csv(index=False), 'Hartford_County_vacinac
```

## Criando Coluna com Dados diários do DataFrame de Hospitalização

### Importando Bibliotecas

```
import os
from google.cloud import storage
import io
from io import BytesIO
```

### Utilizando a chave de Serviço GCP

```
# Criando o client object
storage_client = storage.Client.from_service_account_json('/content/projeto-finalesc01-b94741ddd471.json')
```

### Acessando o Bucket do Projeto

```
BUCKET_NAME = 'projeto-finalesc01'
bucket = storage_client.get_bucket(BUCKET_NAME)
```

### Listando Arquivos do Bucket

```
filename = list(bucket.list_blobs(prefix=''))
for name in filename:
    print(name.name)
```

### Carregando Datasets

```
# Carregando Datasets CSV New_Haven_County
blob = bucket.blob('dados-tratados/New_Haven_County_Hospitalizacoes.csv')
data = blob.download_as_string()

# Carregando Datasets CSV Fairfield_County
blob = bucket.blob('dados-tratados/Fairfield_County_Hospitalizacoes.csv')
data2 = blob.download_as_string()

# Carregando Datasets CSV Hartford_County
blob = bucket.blob('dados-tratados/Hartford_County_Hospitalizacoes.csv')
data3 = blob.download_as_string()
```

## Transformando e Carregando Dados com Pandas

### Importando Pandas

```
import pandas as pd
```

### Criando DataFrame

```
df1 = pd.read_csv(io.BytesIO(data), encoding='utf-8', sep=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)  
df1
```

### Droppando colunas

```
df1.drop(['Casos_Provaveis', 'Casos_Total', 'Casos_Provaveis', 'Casos_Total_Taxa', 'Obitos_Total', 'Obitos_Provaveis', 'Condado_Codigo'], axis=1, inplace=True)
```

### Criando colunas com casos diários

```
df1['Casos_Novos'] = df1['Casos_Confirmados'].sub(df1['Casos_Confirmados'].shift())  
df1['Casos_Novos'].iloc[0] = df1['Casos_Confirmados'].iloc[0]
```

```
df1['Novos_Obitos_Confirmados'] = df1['Obitos_Confirmados'].sub(df1['Obitos_Confirmados'].shift())  
df1['Novos_Obitos_Confirmados'].iloc[0] = df1['Obitos_Confirmados'].iloc[0]
```

### Renomeando colunas

```
df1.rename(columns={'Casos_Confirmados': 'Total_Casos_Confirmados', 'Obitos_Confirmados': 'Total_Obitos_Confirmados', 'Casos_Hospitalizados': 'Total_Casos_Hospitalizados'}, inplace=True)
```

### Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finais01')  
bucket.blob('dados_tratados_condados/New_Haven_County_Hospitalizados.csv').upload_from_string(df1.to_csv(index=False), 'New_Haven_County_Hospitalizados.csv')  
df1.to_csv('New_Haven_County_Hospitalizados.csv', index=False)
```

## Tratamento DataFrame 2

### Criando DataFrame

```
df2 = pd.read_csv(io.BytesIO(data2), encoding='utf-8', sep=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)  
df2
```

### Droppando colunas

```
df2.drop(['Casos_Provaveis', 'Casos_Total', 'Casos_Provaveis', 'Casos_Total_Taxa', 'Obitos_Total', 'Obitos_Provaveis', 'Condado_Codigo'], axis=1, inplace=True)
```

### Criando colunas com casos diários

```
df2['Casos_Novos'] = df2['Casos_Confirmados'].sub(df2['Casos_Confirmados'].shift())
df2['Casos_Novos'].iloc[0] = df2['Casos_Confirmados'].iloc[0]

df2['Novos_Obitos_Confirmados'] = df2['Obitos_Confirmados'].sub(df2['Obitos_Confirmados'].shift())
df2['Novos_Obitos_Confirmados'].iloc[0] = df2['Obitos_Confirmados'].iloc[0]
```

## Renomeando colunas

```
df2.rename(columns={'Casos_Confirmados': 'Total_Casos_Confirmados', 'Obitos_Confirmados': 'Total_Obitos_Confirmados', 'Casos_Hospitalizados'
```

## Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finaiscc01')
bucket.blob('dados_tratados_condados/Fairfield_County_Hospitalizados.csv').upload_from_string(df2.to_csv(index=False), 'Fairfield_County_Hos
df2.to_csv('Fairfield_County_Hospitalizados.csv', index=False)
```

## Tratamento DataFrame 3

### Criando DataFrame

```
df3 = pd.read_csv(io.BytesIO(data3), encoding='utf-8', sep=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)
df3
```

### Droppando colunas

```
df3.drop(['Casos_Provaveis', 'Casos_Total', 'Casos_Provaveis', 'Casos_Total_Taxa', 'Obitos_Total', 'Obitos_Provaveis', 'Condado_Codigo'], axis=1)
```

### Criando coluna com casos diários

```
df3['Casos_Novos'] = df3['Casos_Confirmados'].sub(df3['Casos_Confirmados'].shift())
df3['Casos_Novos'].iloc[0] = df3['Casos_Confirmados'].iloc[0]

df3['Novos_Obitos_Confirmados'] = df3['Obitos_Confirmados'].sub(df3['Obitos_Confirmados'].shift())
df3['Novos_Obitos_Confirmados'].iloc[0] = df3['Obitos_Confirmados'].iloc[0]
```

## Renomeando colunas

```
df3.rename(columns={'Casos_Confirmados': 'Total_Casos_Confirmados', 'Obitos_Confirmados': 'Total_Obitos_Confirmados', 'Casos_Hospitalizados'
```

## Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finaiscc01')
bucket.blob('dados_tratados_condados/Hartford_County_Hospitalizados.csv').upload_from_string(df3.to_csv(index=False), 'Hartford_County_Hosp
df3.to_csv('Hartford_County_Hospitalizados.csv', index=False)
```

## Unindo DataFrames Hospitalizados/Óbitos com Append

```
dfa = df1.append([df2, df3], ignore_index=True)
```

Salvando arquivos e fazendo upload me nuvem

```
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados_tratados_condados/Condados_Hospitalizados.csv').upload_from_string(dfa.to_csv(index=False), 'Condados_Hospitalizados.csv'
dfa.to_csv('Condados_Hospitalizados.csv', index=False)
```

## Criando coluna com Dados diários do DataFrame de Vacinação

Listando Arquivos do Bucket

```
filename = list(bucket.list_blobs(prefix=''))
for name in filename:
    print(name.name)
```

Carregando Datasets Vacinação

```
# Carregando Datasets CSV New_Haven_County Vacinação
blop = bucket.blob('dados-tratados/New_Haven_County_vacinacao.csv')
data4 = blop.download_as_string()

# Carregando Datasets CSV Fairfield_County Vacinação
blop = bucket.blob('dados-tratados/Fairfield_County_vacinacao.csv')
data5 = blop.download_as_string()

# Carregando Datasets CSV Hartford_County Vacinação
blop = bucket.blob('dados-tratados/Hartford_County_vacinacao.csv')
data6 = blop.download_as_string()
```

Tratamento DataFrame 4

Criando DataFrame

```
df4 = pd.read_csv(io.BytesIO(data4), encoding='utf-8', sep=',')
df4.head()
```

Criando colunas com dados diários

```
df4['Vacinias_Dia'] = df4['Numero_total_vacinados'].sub(df4['Numero_total_vacinados'].shift())
df4['Vacinias_Dia'].iloc[0] = df4['Numero_total_vacinados'].iloc[0]

df4['Vacinias_Dia_65+'] = df4['Numero_total_65+'].sub(df4['Numero_total_65+'].shift())
df4['Vacinias_Dia_65+'].iloc[0] = df4['Numero_total_65+'].iloc[0]
```

Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados_tratados_condados/New_Haven_County_Vacinacao.csv').upload_from_string(df4.to_csv(index=False), 'New_Haven_County_Vacinaca
df4.to_csv('New_Haven_County_Vacinacao.csv', index=False)
```

## Tratamento DataFrame 5

### Criando DataFrame

```
df5 = pd.read_csv(io.BytesIO(data5), encoding='utf-8', sep=',')
df5.head()
```

### Criando colunas com dados diários

```
df5['Vacinias_Dia'] = df5['Numero_total_vacinados'].sub(df5['Numero_total_vacinados'].shift())
df5['Vacinias_Dia'].iloc[0] = df5['Numero_total_vacinados'].iloc[0]

df5['Vacinias_Dia_65+'] = df5['Numero_total_65+'].sub(df5['Numero_total_65+'].shift())
df5['Vacinias_Dia_65+'].iloc[0] = df5['Numero_total_65+'].iloc[0]
```

### Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados_tratados_condados/Fairfield_County_Vacinacao.csv').upload_from_string(df5.to_csv(index=False), 'Fairfield_County_Vacinacao.csv')

df5.to_csv('Fairfield_County_Vacinacao.csv', index=False)
```

## Tratamento DataFrame 6

### Criando DataFrame

```
df6 = pd.read_csv(io.BytesIO(data6), encoding='utf-8', sep=',')
df6.head()
```

### Criando colunas com dados diários

```
df6['Vacinias_Dia'] = df6['Numero_total_vacinados'].sub(df6['Numero_total_vacinados'].shift())
df6['Vacinias_Dia'].iloc[0] = df6['Numero_total_vacinados'].iloc[0]

df6['Vacinias_Dia_65+'] = df6['Numero_total_65+'].sub(df6['Numero_total_65+'].shift())
df6['Vacinias_Dia_65+'].iloc[0] = df6['Numero_total_65+'].iloc[0]
```

### Salvando arquivos e fazendo upload em nuvem

```
bucket = storage_client.get_bucket('projeto-finais01')
bucket.blob('dados_tratados_condados/Hartford_County_Vacinacao.csv').upload_from_string(df6.to_csv(index=False), 'Hartford_County_Vacinacao.csv')

df6.to_csv('Hartford_County_Vacinacao.csv', index=False)
```

## Unindo DataFrames de Vacinação com Append

```
dfb = df4.append([df5, df6], ignore_index=True)
```

### Salvando arquivos e fazendo upload em nuvem

```
dfb.drop(['indice'], axis=1, inplace=True)

bucket = storage_client.get_bucket('projeto-finaiscc01')
bucket.blob('dados_tratados_condados/Condados_Vacinacao.csv').upload_from_string(dfb.to_csv(index=False), 'Condados_Vacinacao.csv')

dfb.to_csv('Condados_Vacinacao.csv', index=False)
```

## PySpark e Spark SQL

### PySpark - Hospitalização

#### Instalando PySpark

```
pip install pyspark
```

#### Importando bibliotecas

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType, FloatType, DateType
from pyspark.sql.window import Window
from pyspark.sql.functions import sum
from pyspark.sql.functions import avg
from pyspark.sql.functions import max
from pyspark.sql.functions import min
```

#### Iniciando sessão spark

```
dfspark1 = spark.read.format('csv').option('header','true').option('inverschema', 'true').option('delimiter', ',').load('/content/Condados_
```

```
dfspark1.printSchema()
dfspark1.show()
```

#### Filtros

```
# 1 - Query para mostrar casos novos maior que 500
dfspark1.select(F.col('Data_Atualizacao'), F.col('Condado'), F.col('Casos_Novos')).filter(F.col('Casos_Novos') > 500).show()
```

```
#2 - Query para mostrar o número total de casos confirmados por Condado
dfspark1.groupBy("Condado").agg(sum("Casos_Novos").alias("Casos_Novos")).show()
```

#### Importando Matplotlib

```
import matplotlib.pyplot as plt
```

#### Plotagem

Criando um gráfico a partir de uma query PySpark. Uma variável é criando para guardar a query pyspark, é criando uma variável para o eixo X e eixo Y, depois a query em PySpark é transformando em DF pandas com a função `.toPandas()` para que possa ser plotado utilizando a biblioteca Matplotlib

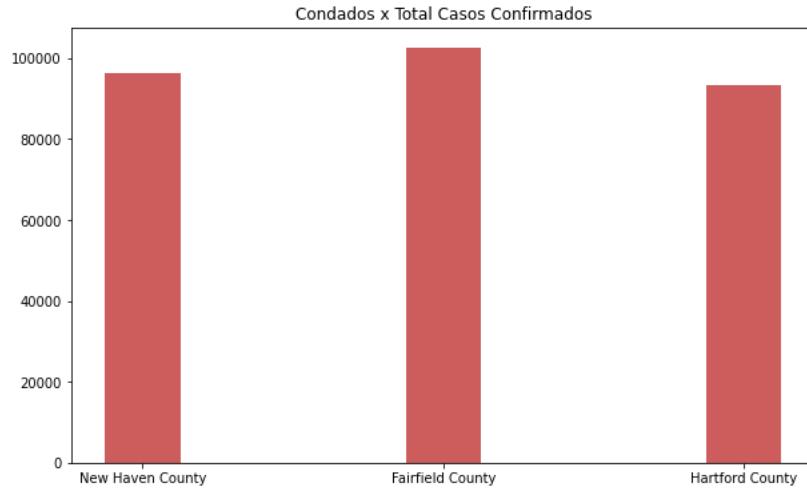
```

dfplot = dfspark1.groupBy('Condado').sum('Casos_Novos')

x = dfplot.toPandas()['Condado'].values.tolist()
y = dfplot.toPandas()['sum(Casos_Novos)'].values.tolist()

plt.figure(figsize=(10, 6))
plt.title('Condados x Total Casos Confirmados')
plt.bar(x,y, color='indianred', width=0.25)
plt.show()

```



```

#3 - Query para verificar a Média de Casos Novos agrupado por Condado
dfspark1.groupBy('Condado').agg(avg('Casos_Novos').alias('Media_Casos_Novos')).show()

```

```

#4 - Query para mostrar Novos Obitos Confirmados maior que 500 por dia
dfspark1.select(F.col('Data_Atualizacao'), F.col('Condado'), F.col('Novos_Obitos_Confirmados')).filter(F.col('Novos_Obitos_Confirmados') > 500).show()

```

```

#5 - Query para verificar a Média de Novos Obitos Confirmados agrupado por Condado
dfspark1.groupBy('Condado').agg(avg('Novos_Obitos_Confirmados').alias('Media_Novos_Obitos_Confirmados')).show()

```

```

#6 - Query para mostrar o número total de Obitos Confirmados por Condado
dfspark1.groupBy("Condado").agg(sum("Novos_Obitos_Confirmados").alias("Soma_Novos_Obitos_Confirmados")).show()

```

## Segunda Plotagem: Condados x Total de Óbitos Confirmados

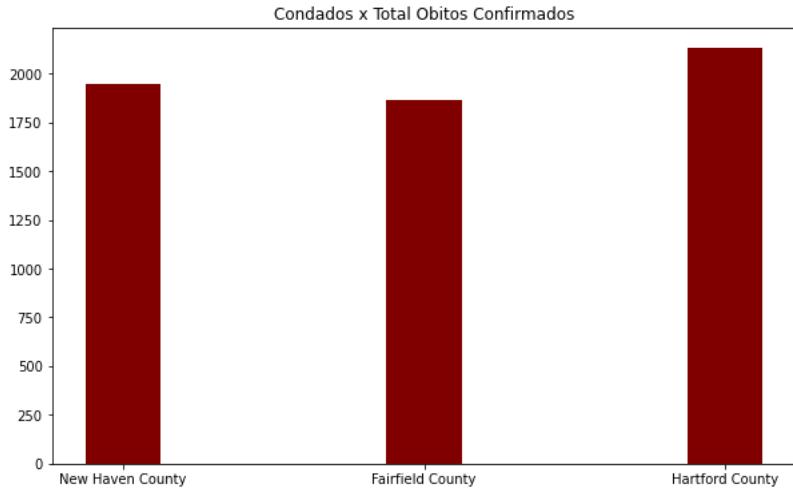
```

dfplot2 = dfspark1.groupBy('Condado').sum('Novos_Obitos_Confirmados')

x2 = dfplot2.toPandas()['Condado'].values.tolist()
y2 = dfplot2.toPandas()['sum(Novos_Obitos_Confirmados)'].values.tolist()

plt.figure(figsize=(10, 6))
plt.title('Condados x Total Obitos Confirmados')
plt.bar(x2,y2, color='maroon', width=0.25)
plt.show()

```



```
#7 - Query para verificar número Máximo de Casos Hospitalizados agrupado por Condado
dfspark1.groupBy('Condado').agg(max('Total_Casos_Hospitalizados').alias('Maximo_Casos_Hospitalizados')).show()
```

## Spark SQL - Hospitalização

### Criando Tabela

```
dfspark1.createOrReplaceTempView('hospitalizacao')
```

### Queries utilizando SQL

```
#1 - visão geral
spark.sql('SELECT * FROM hospitalizacao').show()
```

```
# 2 - Query utilizada para filtrar apenas Condados onde os Casos Novos ultrapasssem 500 */
spark.sql('SELECT Data_Atualizacao, Condado, Casos_Novos FROM hospitalizacao WHERE Casos_Novos > 500').show()
```

```
#3 - Comando utilizado para Somar os Casos Confirmados*/
spark.sql('SELECT Condado, SUM(Casos_Novos) AS Soma_Casos_Novos FROM hospitalizacao GROUP BY Condado ORDER BY Soma_Casos_Novos ASC').show()
```

```
#4 - Comando utilizado para fazer a média de Casos Hospitalizados por Condado*/
spark.sql('SELECT Condado, ROUND(AVG(Casos_Novos), 2) AS Media_Casos_Novos FROM hospitalizacao GROUP BY Condado ORDER BY Media_Casos_Novos')
```

```
#5 - Query utilizada para filtrar apenas Condados onde os Novos Obitos ultrapasssem 500 por dia
spark.sql('SELECT Data_Atualizacao, Condado, Novos_Obitos_Confirmados FROM hospitalizacao WHERE Novos_Obitos_Confirmados > 500').show()
```

```
#6 - Query utilizada para fazer a média de óbitos confirmados por Condado*/  
spark.sql('SELECT Condado, ROUND(AVG(Novos_Obitos_Confirmados), 2) AS Media_Novos_Obitos_Confirmados FROM hospitalizacao GROUP BY Condado ORDER BY Media_Novos_Obitos_Confirmados DESC').show()
```

```
#7 - Query utilizada para Somar os Óbitos Confirmados*/  
spark.sql('SELECT Condado, SUM(Novos_Obitos_Confirmados) AS Soma_Novos_Obitos_Confirmados FROM hospitalizacao GROUP BY Condado ORDER BY Soma_Novos_Obitos_Confirmados DESC').show()
```

```
#8 - Cria uma tabela com o máximo Casos Confirmados, Hospitalizados e a média dos Casos Novos */  
spark.sql('SELECT Condado, MAX(Total_Casos_Confirmados) AS Max_Casos_Confirmados, MAX(Total_Casos_Hospitalizados) AS Max_Casos_Hospitalizados, ROUND(AVG(Casos_Novos), 2) AS Media_Casos_Novos FROM hospitalizacao GROUP BY Condado ORDER BY Media_Casos_Novos DESC').show()
```

```
#9 - Cria um Query com Soma de Casos Novos, Somas de Óbitos Novos e suas Médias, agrupadas por Condado  
spark.sql('SELECT Condado, SUM(Casos_Novos) AS Soma_Casos_Novos, SUM(Novos_Obitos_Confirmados) AS Soma_Obitos_Confirmados, ROUND(AVG(Casos_Novos), 2) AS Media_Novos_Obitos_Confirmados FROM hospitalizacao GROUP BY Condado ORDER BY Media_Casos_Novos DESC').show()
```

## PySpark - Vacinação

### Criando DataFrame

```
dfspark2 = spark.read.format('csv').option('header', 'true').option('inferSchema', 'true').option('delimiter', ',').load('/content/Condados_Vacinas.csv')  
  
dfspark2.printSchema()  
dfspark2.show()
```

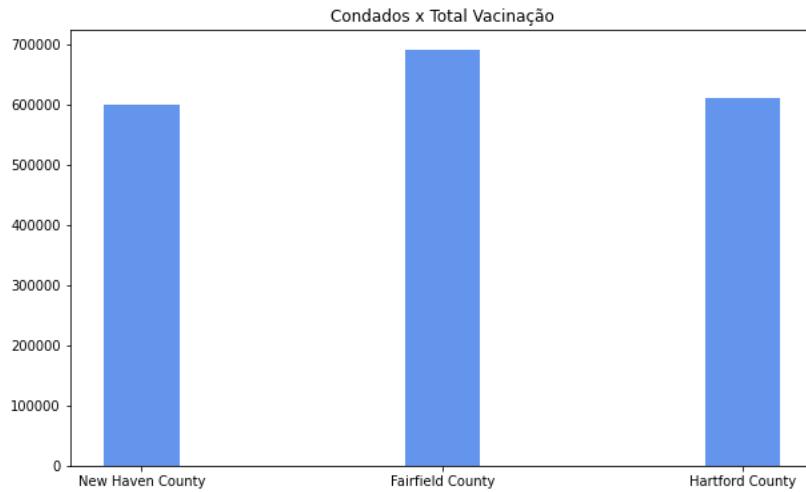
### Filtros

```
# 1 - Query para mostrar Vacinação por dia maior que 500  
dfspark2.select(F.col('Data'), F.col('Condado'), F.col('Vacinas_Dia')).filter(F.col('Vacinas_Dia') > 500).show()
```

```
#2 - Query para mostrar o número total de Vacinação por Condado  
dfspark2.groupBy("Condado").agg(sum("Vacinas_Dia").alias("Total_Vacinacao")).show()
```

### Plotagem com Matplotlib a partir de Query em PySpark

```
dfplot3 = dfspark2.groupBy('Condado').sum('Vacinas_Dia')  
  
x3 = dfplot3.toPandas()['Condado'].values.tolist()  
y3 = dfplot3.toPandas()['sum(Vacinas_Dia)'].values.tolist()  
  
plt.figure(figsize=(10, 6))  
plt.title('Condados x Total Vacinação')  
plt.bar(x3,y3, color='cornflowerblue', width=0.25)  
plt.show()
```



```
# 3 - Query para mostrar Vacinação 65 por dia 200
dfspark2.select(F.col('Data'), F.col('Condado'), F.col('Vacinas_Dia_65+')).filter(F.col('Vacinas_Dia_65+') < 200).show(124)
```

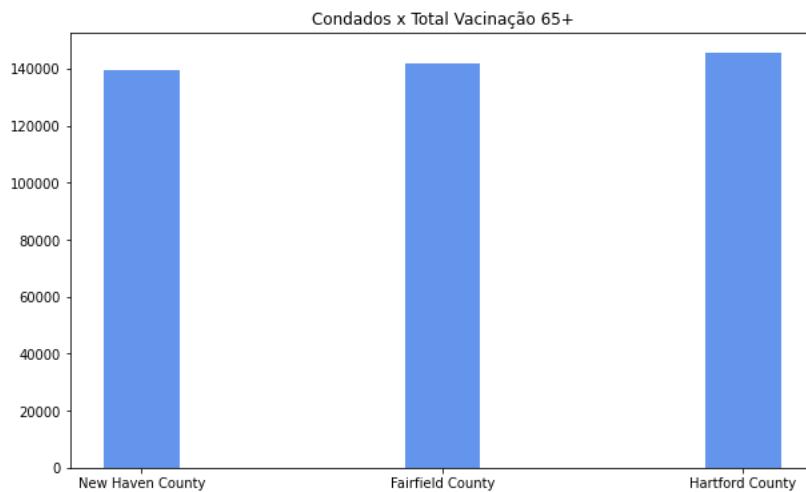
```
#4 - Query para mostrar o número total de Vacinação 65+ por Condado
dfspark2.groupBy("Condado").agg(sum('Vacinas_Dia_65+').alias('Total_Vacinas_65+')).show()
```

#### Plotagem: Condados X Total de Vacinação 65+

```
dfplot4 = dfspark2.groupBy('Condado').sum('Vacinas_Dia_65+')

x4 = dfplot4.toPandas()['Condado'].values.tolist()
y4 = dfplot4.toPandas()['sum(Vacinas_Dia_65+)'].values.tolist()

plt.figure(figsize=(10, 6))
plt.title('Condados x Total Vacinação 65+')
plt.bar(x4,y4, color='cornflowerblue', width=0.25)
plt.show()
```



```
#5 - Query para verificar Porcentagem Máxima de Vacinação 65+ agrupado por Condado  
dfspark2.groupBy('Condado').agg(max('Porcentagem_total_65+').alias('Maximo_Porcentagem_total_65+')).show()
```

## Spark SQL - Vacinação

Renomeando colunas que causam conflito na query

```
dfspark2 = dfspark2.withColumnRenamed('Numero_total_65+', 'Numero_total_65')  
dfspark2 = dfspark2.withColumnRenamed('Porcentagem_total_65+', 'Porcentagem_total_65')  
dfspark2 = dfspark2.withColumnRenamed('Vacinias_Dia_65+', 'Vacinias_Dia_65')
```

## Criando Tabela

```
dfspark2.createOrReplaceTempView('vacinacao')
```

## Utilizando SQL

```
#1 - visão geral  
spark.sql('SELECT * FROM vacinacao').show()
```

```
#2 - Query utilizada para visualizar o número de vacinações por dia maiores de 500 */  
spark.sql('SELECT Data, Condado, Vacinas_Dia FROM vacinacao WHERE Vacinas_Dia > 500').show()
```

```
#3 - Query utilizada para mostrar a soma de vacinação por condados */  
spark.sql('SELECT Condado, SUM(Vacinias_Dia) AS Soma_Vacinias_Dia FROM vacinacao GROUP BY Condado ORDER BY Soma_Vacinias_Dia ASC').show()
```

```
#4 - Query utilizada para visualizar o número de idosos vacinados menores que 200 */  
spark.sql('SELECT Data, Condado, Vacinas_Dia_65 FROM vacinacao WHERE Vacinas_Dia_65 < 200').show()
```

```
#5 - Query para verificar Porcentagem Máxima de Vacinação 65+ agrupado por Condado */  
spark.sql('SELECT Condado, MAX(Porcentagem_total_65) AS Porcentagem_Total_65_Maxima FROM vacinacao GROUP BY Condado ORDER BY Porcentagem_Tot
```

## Plotagens de Série Histórica com Seaborn

Seaborn é uma biblioteca de visualização de dados Python baseada em matplotlib. Ele fornece uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos.

## Importando Bibliotecas

```
import seaborn as sns  
import pandas as pd  
import matplotlib  
import matplotlib.pyplot as plt
```

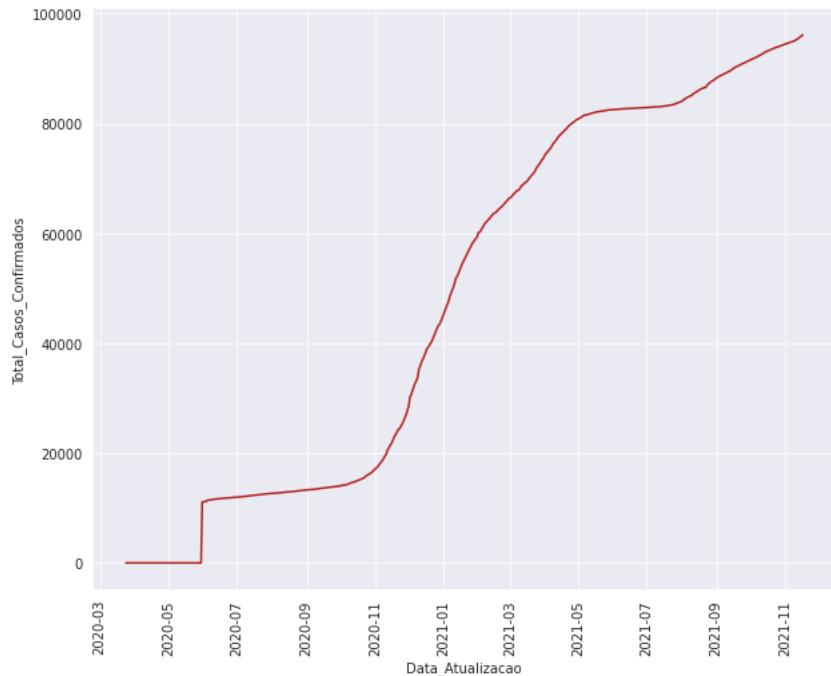
## Criando DataFrames

```
df_nhc = pd.read_csv('/content/New_Haven_County_Hospitalizados.csv', delimiter=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)
df_fc = pd.read_csv('/content/Fairfield_County_Hospitalizados.csv', delimiter=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)
df_hc = pd.read_csv('/content/Hartford_County_Hospitalizados.csv', delimiter=',', parse_dates = ['Data_Atualizacao'], dayfirst = True)
```

## New Haven County

### Plotagem da Série Histórica do número de casos

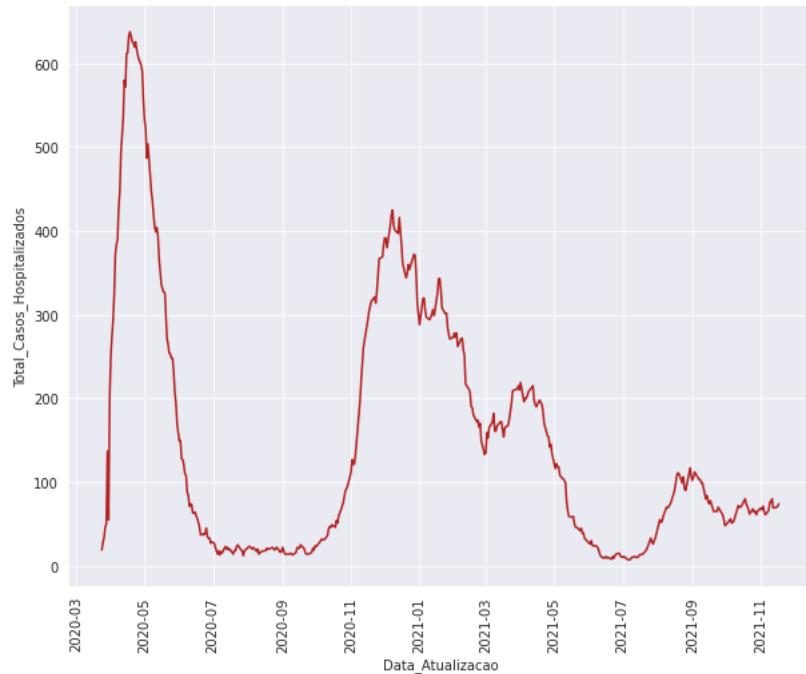
```
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_nhc, x='Data_Atualizacao', y='Total_Casos_Confirmados', color='firebrick')
plt.xticks(rotation=90)
```



A partir desse gráfico é possível analisar os picos/ondas de casos e relacioná-los as variáveis do vírus

### Plotagem do número de hospitalizados

```
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_nhc, x='Data_Atualizacao', y='Total_Casos_Hospitalizados', color='firebrick')
plt.xticks(rotation=90)
```



A partir desse gráfico é possível analisar os picos/ondas de hospitalização e a diminuição que está relacionada com o processo de vacinação que reduziu o numero de casos hospitalizados.

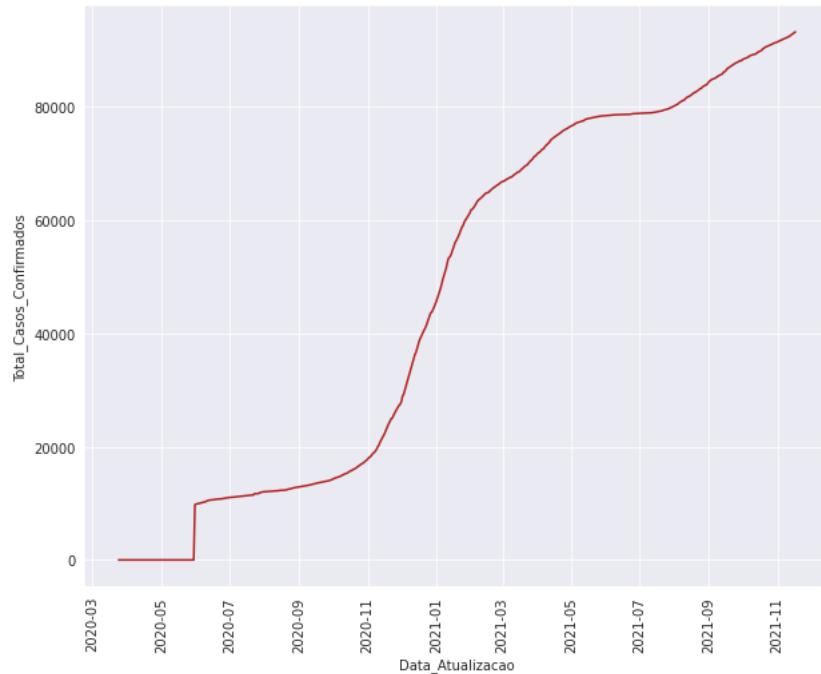
Hartford County

Plotagem da Série Histórica do número de casos

```

sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_hc, x='Data_Atualizacao', y='Total_Casos_Confirmados', color='firebrick')
plt.xticks(rotation=90)

```



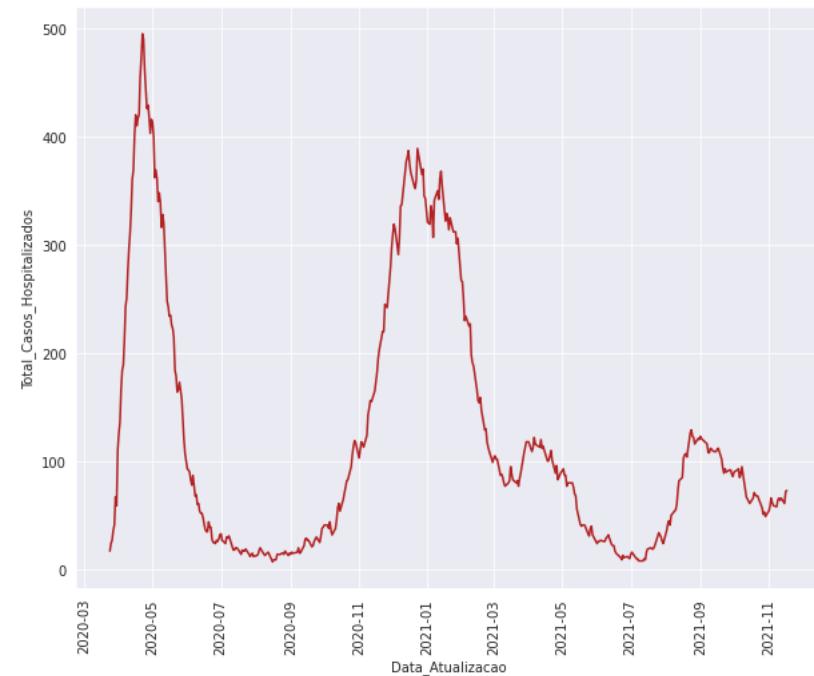
A partir desse gráfico é possível analisar os picos/ondas de casos e relacioná-los as variáveis do vírus

#### Plotagem do número de hospitalizados

```

sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_hc, x='Data_Atualizacao', y='Total_Casos_Hospitalizados', color='firebrick')
plt.xticks(rotation=90)

```

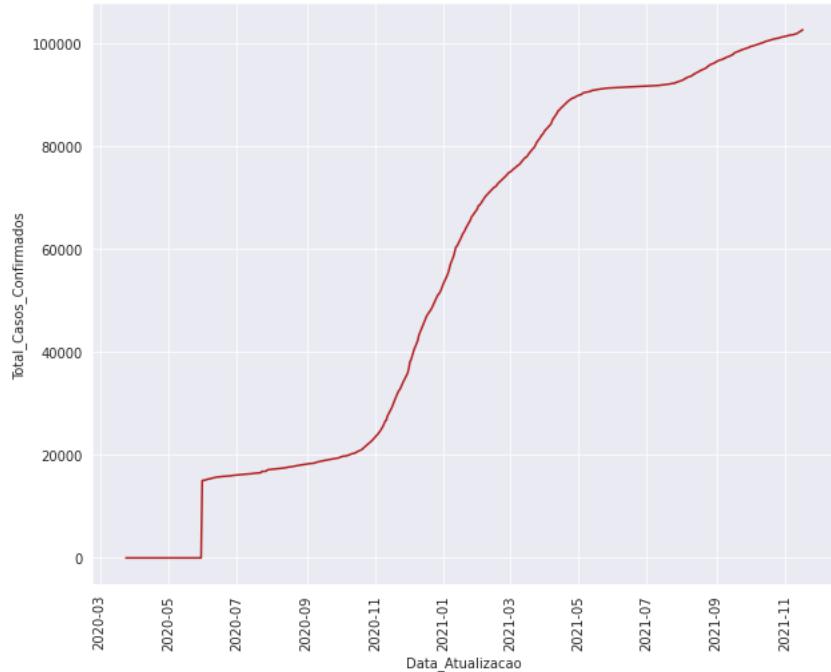


A partir desse gráfico é possível analisar os picos/ondas de hospitalização e a diminuição que está relacionada com o processo de vacinação que reduziu o numero de casos hospitalizados.

#### Farfield County

##### Plotagem do número de casos

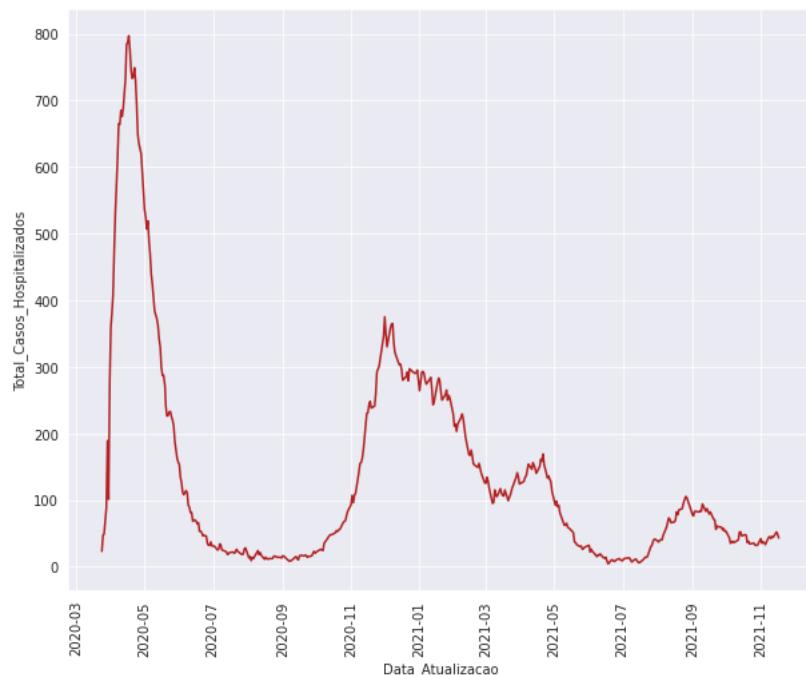
```
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_fc, x='Data_Atualizacao', y='Total_Casos_Confirmados', color='firebrick')
plt.xticks(rotation=90)
```



A partir desse gráfico é possível analisar os picos/ondas de casos e relacioná-los as variáveis do vírus

##### Plotagem do número de hospitalizados

```
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
sns.lineplot(data= df_fc, x='Data_Atualizacao', y='Total_Casos_Hospitalizados', color='firebrick')
plt.xticks(rotation=90)
```



A partir desse gráfico é possível analisar os picos/ondas de hospitalização e a diminuição que está relacionada com o processo de vacinação que reduziu o numero de casos hospitalizados.