Instituto Federal de Brasília

Campus Brasília

Curso Superior de Tecnologia em Sistemas para Internet

RELATÓRIO - PRÁTICA INTEGRADA ICD E IIA – G2 – EFLM

Por

ELIZEU GROSSKOPF SCHLOTTFELDT NETO

FRANKLIN HUDSON OLIVEIRA LIMA

LUCAS BARROS DE ARAÚJO

MÔNICA DE OLIVEIRA

BRASÍLIA

2020

Sumário

1. **Objetivos**
    a. **Coleta ----------p. 3**
    b. **Exploração ---------- p. 4**
    c. **Exploração com gráficos e mapas ---------- p. 5**
    d. **Limpeza dos dados ---------- p. 7**
    e. **Acréscimo de variáveis ---------- p. 8**
    f. **Dados no MongoDB ---------- p. 9**
    g. **Análise Temporal ---------- p.11**
2. **Desenvolvimento**
    a. **Código implementado ----------p. 14**
3. **Considerações Finais ----------p. 34**
4. **Referências ----------p. 35**

1. Objetivos

Coleta

Inicialmente, nesta etapa do projeto, eu tive que desenvolver um script em Python para realizar a coleta de dados do site da "*NUFORC – National UFO Reporting Center*".

Foram coletados dados de um período de 20 anos, entre setembro de 1997 até agosto de 2017. Utilizei as bibliotecas: "*requests*" (para execução de requisições HTTP), "*BeautifulSoup*" (para extração de dados em arquivos HTML e XML), e a "*Pandas*" (para armazenar, limpar e salvar os dados em forma de tabela).

No final, consegui realizar a raspagem da web (web scraping), salvando os dados da coleta em um arquivo csv chamado de "ovnis_tabela.csv".

O conjunto de dados a ser estudado possui relatórios acerca dos OVNIS avistados pela NUFORC. Os atributos que compõem esses dados são:

- "DATE/TIME" data e dia, de quando que o ovni foi avistado;
- "CITY" cidade onde foi visto o óvni;
- "STATE" estado onde o ovni foi avistado;
- "SHAPE" formato do óvni;
- "DURATION" a duração da aparição do óvni;
- "SUMMARY" descrição da aparição do ovni de forma detalhada;
- "POSTED" a data de postagem no site, do item identificado.

Essas informações são fornecidas pela NUFORC. Um estudo minucioso com esses dados será realizado, a fim de investigar perguntas relevantes sobre as aparições dessas aeronaves alienígenas, dessa forma alcançando conclusões.

Exploração

Na etapa de exploração do projeto, nós continuamos a trabalhar com o arquivo csv chamado de “ovnis_tabela.csv”.

A biblioteca “*pandas*” foi utilizada para a análise dos dados, nós realizamos algumas operações utilizando alguns métodos dessa biblioteca para respondermos algumas perguntas:

Saber a quantidade de linhas, observações ou variáveis que foram coletadas.

Quantos relatos ocorreram por estado em ordem decrescente?

Remover possíveis campos vazios (sem estado).

Limitar a análise aos estados dos Estados Unidos.

Consulta por cidades, com o objetivo de saber quais contêm o maior número de relatos (cidades que apresentem ao menos 10 relatos).

Com o dado anterior, responder a seguinte pergunta: por que será que essa é a cidade que possui mais relatos?

Fazer uma query exclusiva para o estado com maior número de relatos, buscando cidades que possuam um número superior a 10 relatórios. Enfatizar a cidade, a quantidade de relatos e formato do objeto não identificado.

No final, após a exploração, salvamos os dados das análises em um arquivo csv chamado de “ovnis_maiores_relatos.csv”.

Exploração com gráficos e mapas

Nesta etapa do problema, fizemos o import e as instalações das bibliotecas pandasql, geopandas, zipcodes, folium e pandas a fim de realizarmos as análises.

Estamos partindo do arquivo criado ‘ovnis_clean.csv’.

Usando o pacote pysqldf que permite consultar DataFrames do pandas usando a sintaxe SQL. O pysqldf tem por objetivo buscar uma maneira mais familiar d e manipular e limpar dados para novos usuários em python ou pandas.

Criamos uma função para calcular quais os quatro estados com maior frequência de relatos e tipos de OVNIS mais populares.. Ele possui uma query e um conjunto de variáveis globais do ambiente.

O resultado da query traz o resultado esperado e a seguir, armazenamos em um DataFrame.

A seguir salvamos o novo conjunto de dados para um arquivo .csv intitulado: ‘df_ovnis_4states.csv’.

Plotamos os gráficos em barras agrupadas de duas formas diferentes: (fig 1. e fig 2.) e em barras empilhadas (fig 3.)

Utilizando os dados que nos seriam úteis da biblioteca zipcodes, atribuimos aos DataFrame ‘df_zipcodes’, fizemos a exclusão dos dados duplicados e salvamos num arquivo csv: ‘zip_code.csv’.

Fizemos a leitura dos dados do arquivo (do sprint anterior) ‘ovnis_clean.csv’, atribuindo seus valores ao data frame ‘df_dados’. Fizemos a alteração da fonte dos dados ‘city, state, lat e long a fim de igualar ao do DataFrame df.zipcodes.csv.

Criamos o DataFrame ‘df_todos_os_dados’ juntando, através de um merge, as latitudes e longitudes encontradas no DataFrame ‘df_zipcodes’ e as cidades encontradas no DataFrame ‘df_dados’ e após, fizemos a seleção dos dados a serem utilizados (city, state, lat e long) e salvamos no DataFrame ‘df_todos_os_dados’. A seguir convertemos para o arquivo .csv ‘dados_para_mapa.csv’.

Neste momento é necessário ler o arquivo novamente a fim de não dar erro e conseguir plotar o mapa.

Criamos um array com a latitude e longitude dos países do DataFrame ‘df_todos os dados’ e fazendo o uso do geopandas, buscamos as coordenadas dos EUA. Passamos estas coordenadas para uma instância de um novo mapa, utilizando o folium. Neste ponto é necessário aumentar o limite de entradas para evitar erros, devido à quantidade de dados utilizados. Usando as coordenadas das cidades, criamos um mapa de calor com estes parâmetros.

Nesta fase, separamos os dados do estado da Califórnia e atribuímos à um DataFrame: ‘df_california’ e com estes dados criamos um array com a latitude e a longitude. Novamente usamos o geopandas, agora para buscar as coordenadas da Califórnia e repetimos o processo de instanciar um mapa usando o folium, mas agora com as coordenadas da Califórnia. Com estes parâmetros, criamos um mapa de calor das cidades da Califórnia.

fig1.

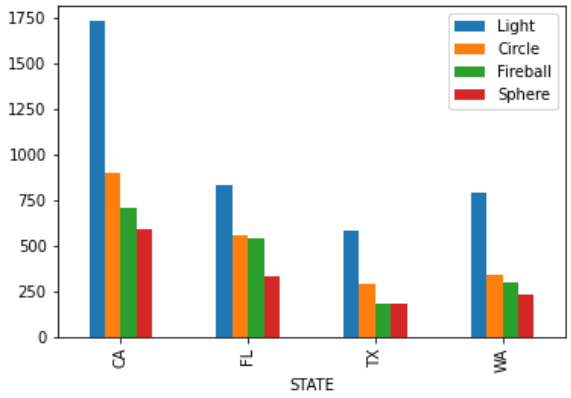


fig2.

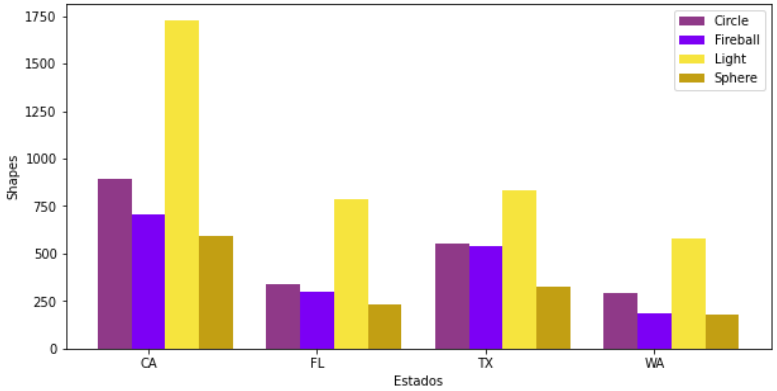
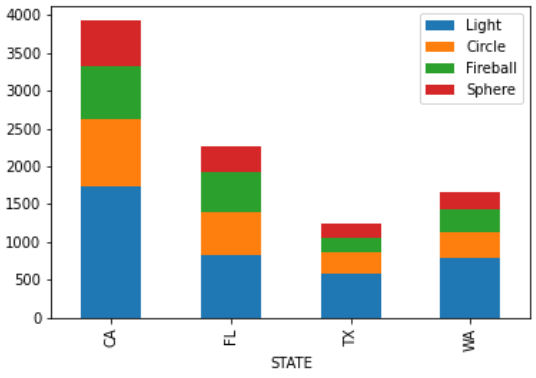


fig 3.



Limpeza dos dados

Nesta fase usamos o arquivo .csv: ‘ovnis_tabela.csv’, criamos o DataFrame ‘df_ovnis’, verificamos as colunas cujos valores são vazios, eliminamos estes valores das colunas ‘city’, ‘state’ e ‘shape’.

Verificamos os valores existentes para a coluna ‘stae’ e selecionamos os estados que NÃO são dos Estados Unidos, e fizemos a remoção destes, salvando estes dados no DataFrame ‘ultimo_df’ e após, enviamos para um arquivo .csv: ‘ovnis_clean.csv’. Descartamos então as colunas ‘duration’, ‘summary’ e ‘posted’.

Fizemos uma verificação de quantos valores haviam para a coluna ‘shape’ e a seguir, que possuíam valores superiores a 1000 então selecionamos e removemos os formatos com as ocorrências abaixo de 1000.

Salvamos os novos conjuntos de dados num arquivo .csv: ‘df_OVNI_limpo.csv.

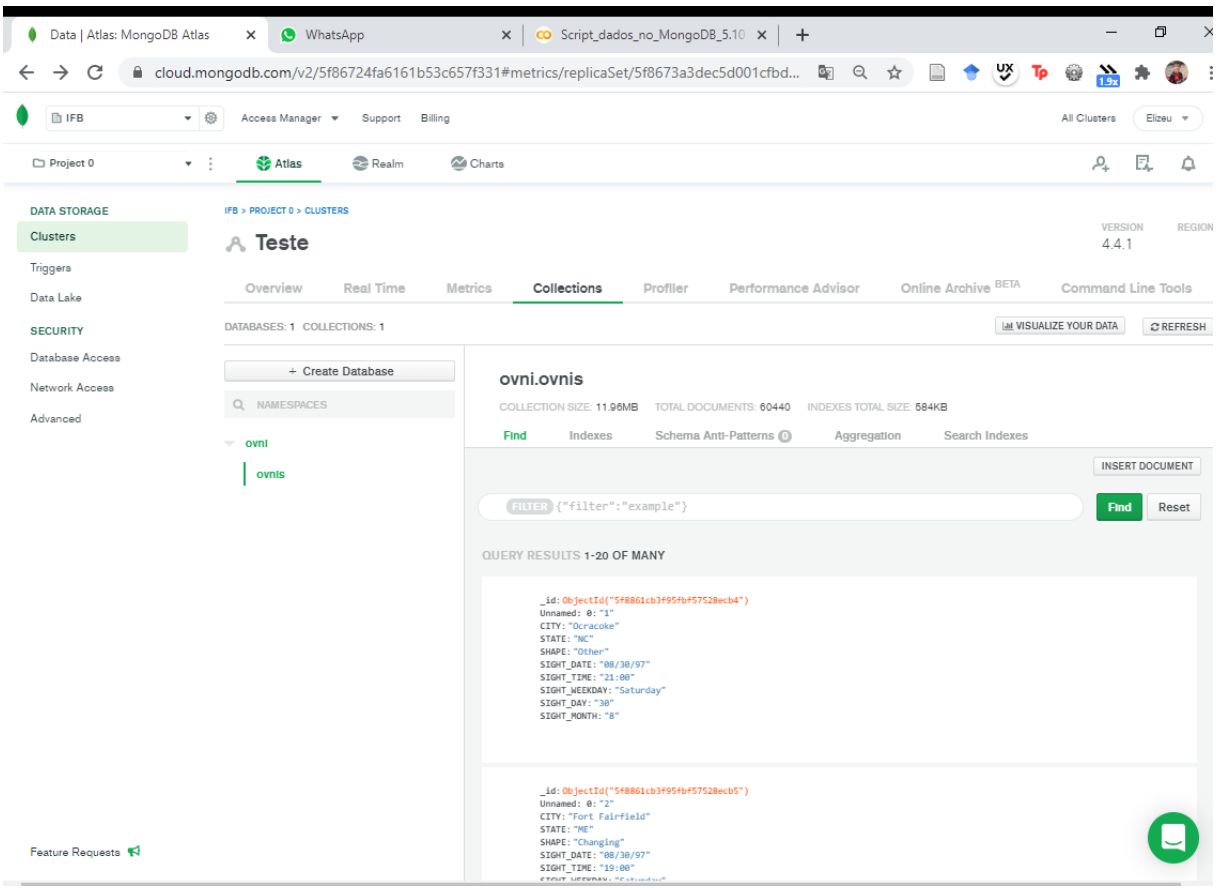
Acréscimo de Variáveis

Primeiro, dividimos o conteúdo da Coluna Date/Time em duas colunas com data e horário separados e deletamos a coluna Date/Time. A seguir criamos uma coluna para separar os dias da semana.

Separamos então o mês e o dia para melhor refino das pesquisas.

Os resultados foram salvos no arquivo .csv ‘df_OVNI-preparado.csv’.

Dados no MongoDB



Após realizar a instalação das bibliotecas: pymongo, dnsPython e os imports, abrimos conexão com Altas: MongoDF - Password: VKmFe5ghPBIIi3K1, criamos um banco de dados chamado ovni e uma coleção chamada ovnis. Fizemos então a leitura do arquivo df_OVNI_preparado.csv para verificarse ele fora importado.

Transformamos o CSV em um dicionário e inserimos na coleção criada todos os registros do df_OVNI_preparado.

Análise

Nesta etapa, contamos e mostramos quantos documentos há na coleção ovnis. A seguir, resgatamos todos os documentos (registros) da coleção ovnis e ordenamos por tipo. Verificamos quantas ocorrências existem por estado e agregamos para contar todas as ocorrências. (usamos a função aggregate() de collections e atribuímos o valor (1 = True) para a operação sum.). Buscamos então, todas as ocorrências da cidade Phoenix.

Buscamos então as ocorrências do estado da Califórnia e ocultamos o id de cada registro. (Usamos o projection da função find() de collections e atribuímos o valor (0 = False).)

Análise Temporal

Ordenamos as observações de forma ascendente temporalmente (da observação mais antiga para a mais recente), criamos um dataframe com os dados da cidade de Phoenix e agrupamos por data. Excluímos as colunas cujos dados não eram necessários.

Atribuímos o dataframe a um outro para manipulação mais livre e definimos o ano que queríamos que fosse visto no gráfico.

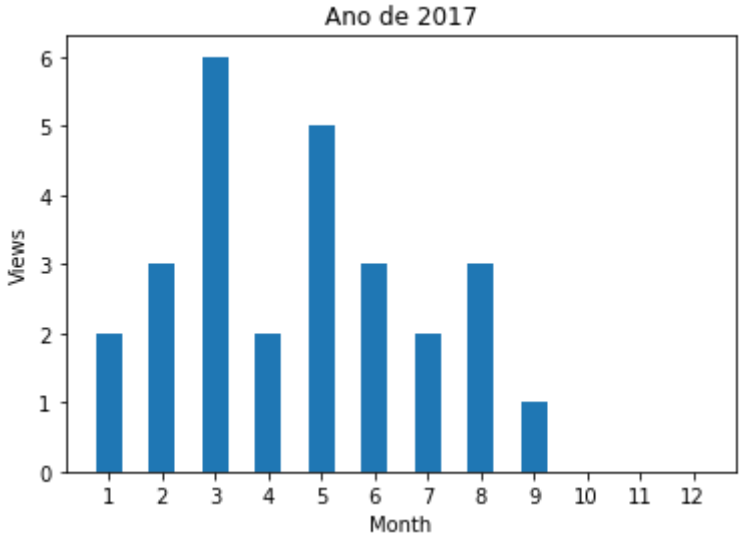
Transformamos a coluna ‘Sight_Date’ em formato de data para separar o ano e separamos o ano em uma coluna própria. Transformamos o ano em uma string e contamos quantas visualizações cada mês tem.
Criamos um vetor com todos os meses do ano e criamos um vetor vazio com as views de cada mês.

No primeiro ‘for’ ele vai rodar todos os 12 meses, colocamos uma flag para não duplicar o número de views de meses repetidos. O segundo ‘for’ irá iterar linha por linha do data frame, atribuímos o número do mês correspondente (representado pelo ‘j’).

Preenchemos os meses cujo ‘n’ tiveram observações registradas com o valor 0.

PLOTANDO O GRÁFICO

Criamos uma distância entre as barras e plotamos as barras, colocamos o nome dos meses como label do eixo x:

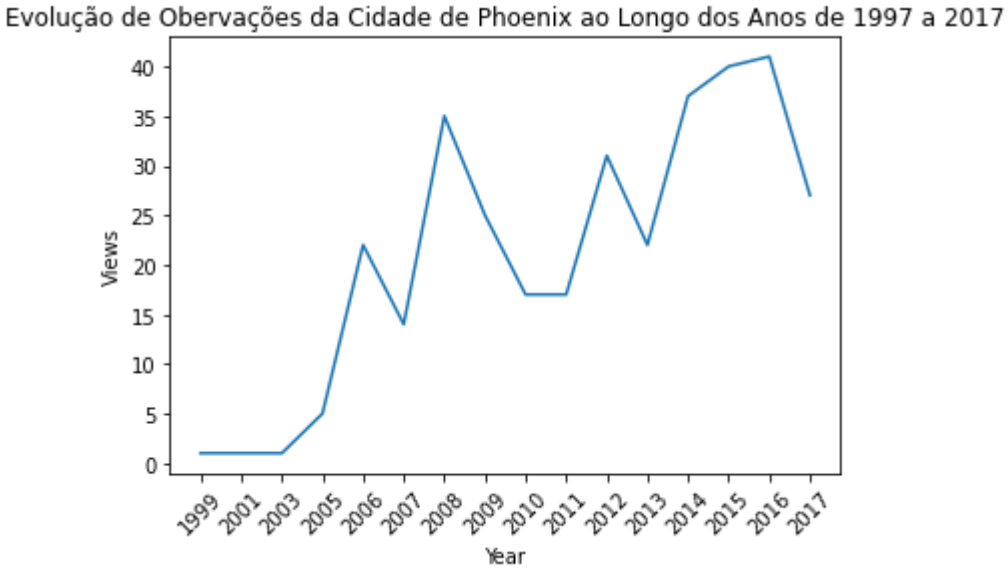


Atribuímos um data frame a outro para manipulação mais livre. Criamos arrays vazios para os anos de 1997 a 2017, outro para as views correspondentes ao período solicitado e outro para os anos que aparecerão no gráfico

Transformamos a coluna ‘Sight_Date’ em formato de data para separar o ano, separamos o ano em uma coluna própria e contamos quantas visualizações cada ano tem.

Iteramos os anos de 1997 a 2017 no array ‘anos’. No primeiro ‘for’ ele vai rodar todos os anos que o array recebeu, criamos uma flag para não duplicar o número de views de anos repetidos e um segundo ‘for’ que irá iterar linha por linha do dataframe. Atribuímos o ano correspondente à view adicionada e o número do ano correspondente (representado pelo j).

Atribuímos os ‘anos_grafico’ como anos do eixo x e as views no eixo y:



Transformamos o ‘Sight_date’ de df_views em tipo Date e definimos o‘Sight_Date’ como o index.

É separado então os conjunto df_views no conjunto de treinamento (df_train) e no conjunto de teste (df_test). O conjunto df_train possui 70% dos dados do conjunto df_views, enquanto o df_test possui 30% dos dados.

Utilizamos o pacote statsmodels e utilizamos a função SARIMAX para criar um modelo.

Acessamos a propriedade AIC para medir a qualidade do modelo ajustado.

Usamos a função forecast sobre o modelo para fazer as previsões sobre a quantidade de dias existentes no conjunto de testes.

Finalmente calculamos o erro médio e o desvio padrão com relação ao conjunto. O erro médio é calculado quando calculamos o valor absoluto da diferença entre o forecast e o df_teste, depois pegamos a média deste vetor resultante.

O método que calcula o desvio padrão da amostra é std(). ddof modifica o divisor da sum dos quadrados das amostras-menos-média. O divisor é N - ddof , onde o padrão ddof é 0 como você podemos ver no resultado.

2. Desenvolvimento

Código implementado

Link do código do script da coleta:

https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20da%20coleta/Script_coleta_5_2.ipynb

```
#Instala as bibliotecas no collab
!pip install requests
!pip install beautifulsoup4

#Realizar os Imports das bibliotecas
import urllib.request
from bs4 import BeautifulSoup

import pandas as pd
import numpy as np

def make_soup(url):
    #   """
     #   Função para adicionar as colunas(features) de uma lista em outr
a lista, na mesma ordem.
     #   INPUT:
     #   url: link do site contendo as tabelas com os reports dos ovnis.
     #   OUTPUT:
     #   soup: site armazenado no formato BeautifulSoup.
     #   """
  #Consulta o site e retorna o html para a variável 'pagina'.
  pagina = urllib.request.urlopen(url)
  #Parse o html na variável 'pagina' e armazena no formato BeautifulSou
p
  soup = BeautifulSoup(pagina, "html.parser")
  return soup

#Array que recebe os urls que desejamos analisar.
urls = []

#Loop que vai percorrer o periodo de setembro 1997 até agosto de 2017.
for y in range(1997,2018):
  for m in range(1,13):
    y2 = y
    if y == 1997 and m < 8:
      continue
    if y == 2017 and m>10:
      break
    m2 = m
    if m<10:
      m2= f'0{m}'
```

```
    urls.append(f'http://www.nuforc.org/webreports/ndxe{y2}{m2}.html')
#Especifica o URL da página que iremos fazer o webscraping

#Loop que printa os urls que vamos coletar os dados.
for x in urls:
  print(x)

#Declaração de um array vazio
array = []

#Loop pra fazer a url
for x in urls:
  #Chama a função make_soup.
  soup = make_soup(x)
  #Procura as 'tr'
  for record in soup.findAll('tr'):
    #Procura as 'td'
    for data in record.findAll('td'):
      #Procura o conteúdo de 'font'
      cols = record.findAll('font')
      #Adiciona as linhas do registro do df.
      array.append((cols[0].text.strip(), cols[1].text.strip(), cols[2]
.text.strip(), cols[3].text.strip(),cols[4].text.strip(),cols[5].text.s
trip(),cols[6].text.strip()))
      break;

#Cria um array numpy.
array2 = np.asarray(array)

#Adiciona os dados em um dataframe.
df = pd.DataFrame(array2)

#Cria as colunas do dataframe.
df.columns = ['DATE/TIME', 'CITY', 'STATE','SHAPE', 'DURATION','SUMMARY
','POSTED']

#Salva o conjunto de dados dataframe em um arquivo csv.
df.to_csv('ovnis_tabela.csv')

#Printa os 10 primeiros elementos do dataframe.
df.head(10)
```

Link do código do script da exploração:

https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20da%20explora%C3%A7%C3%A3o/Script_explora%C3%A7%C3%A3o_5_4.ipynb

```
# Os imports, que vamos usar para realizarmos as análises
```

```
# também pode-se usar outras bibliotecas se desejar.
import pandas as pd
import numpy as np

# Abra o arquivo CSV e mostre algumas linhas
df_ovnis = pd.read_csv('ovnis_tabela.csv')
df_ovnis.head()

df_ovnis.info()

df_ovnis['STATE']
#descobre todos os estados possiveis
df_ovnis['STATE'].unique()

df_ovnis['STATE'].value_counts()

# exiba a contagem de valores faltantes para cada atributo em OVNIS
df_ovnis.isnull().sum()

#Remove os campos vazios que não possuem estado
df_ovnis.dropna(subset=['STATE'], inplace=True)

df_ovnis.isnull().sum()

# salva os novos conjuntos de dados para a próxima seção
df_ovnis.to_csv('ovnis_clean.csv', index=False)

# abre o novo conjunto de dados que foi criado após realizarmos a limpe
za dos dados
df_ovnis = pd.read_csv('ovnis_clean.csv')

df_ovnis.info()

df_ovnis['STATE'].unique()

# seleciona os estados que não são dos Estados Unidos
df_remove = df_ovnis.loc[(df_ovnis['STATE'] == 'NS') | (df_ovnis['STATE
'] == 'QC') | (df_ovnis['STATE'] == 'ON') | (df_ovnis['STATE'] == 'BC')
 | (df_ovnis['STATE'] == 'AB') | (df_ovnis['STATE'] == 'NT') | (df_ovni
s['STATE'] == 'NB') | (df_ovnis['STATE'] == 'PE') | (df_ovnis['STATE']
== 'SK') | (df_ovnis['STATE'] == 'SA') | (df_ovnis['STATE'] == 'MB') |
(df_ovnis['STATE'] == 'NF') | (df_ovnis['STATE'] == 'YT')]
df_remove.head()
ultimo_df = df_ovnis.drop(df_remove.index)

# verifica se conseguiu realizar a limpeza
ultimo_df.info()

# segunda verificação
ultimo_df['STATE'].unique()

# salva os novos conjuntos de dados para a próxima seção
ultimo_df.to_csv('ovnis_clean.csv', index=False)
```

```
# abre o novo conjunto de dados que foi criado após realizarmos a limpe
za dos dados
df_ovnis = pd.read_csv('ovnis_clean.csv')

df_ovnis['CITY'].value_counts()

df_ovnis['CITY'].value_counts() >= 10

# seleciona as cidades que possuem o maior número de relatos.
df_cidades_mr = df_ovnis['CITY'].value_counts() >= 10
df_cidades_mr = df_ovnis.groupby('CITY').count()
df_cidades_mr = df_cidades_mr.sort_values(ascending=False, by="POSTED")
df_cidades_mr = df_cidades_mr.query('POSTED >= 10')['POSTED']

df_cidades_mr

#for i in df_cidades_mr:
#    print(i)
#if df_cidades_mr. == True:
  #print(df_cidades_mr.attrs)
```

Phoenix é a cidade que possui mais relatos. Porque é uma cidade com muitas bases aéreas, as pessoas achavam que viam ovnis quando na realidade estavam vendo aeronaves.

```
# Nós vimos na pergunta 2 que o estado com mais casos era a "CA", calif
órnia.
state_california = df_ovnis[df_ovnis['STATE']=='CA']

COLUNAS = [
    'CIDADE',
    'QUANTIDADE',
    'FORMATO'
]

df_final = pd.DataFrame(columns=COLUNAS)
df_final['CIDADE'] = state_california['CITY']
df_final['QUANTIDADE'] = state_california.groupby('CITY')['CITY'].trans
form('count')
df_final['FORMATO'] = state_california['SHAPE']

#gera arquivo .csv
df_final.to_csv("ovnis_maiores_relatos.csv")

df_final
```

Link do código do script ‘Exploração com gráficos e mapas’:

https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20explora%C3%A7%C3%A3o%20com%20gr%C3%A1ficos%20e%20mapas/Script_explora%C3%A7%C3%A3o_com_gr%C3%A1ficos_e_mapas_5_5.ipynb

```
# Os imports e instalações, que vamos usar para realizarmos as análises
# também pode-se usar outras bibliotecas se desejar.
!pip install pandasql
!pip install geopandas
!pip install zipcodes
!pip install folium
!pip install pandas

import pandas as pd
import numpy as np
import pandasql
from pandasql import sqldf
import matplotlib.pyplot as plt
import geopandas as gpds
import zipcodes as zpc
import sys
import folium
from folium import plugins

# Abra o arquivo CSV e mostre algumas linhas
df_ovnis = pd.read_csv('ovnis_clean.csv')
df_ovnis.head()
# pysqldf permite consultar DataFrames do pandas usando a sintaxe SQL. Funciona de forma semelhante ao sqldf em R.
# O pysqldf busca fornecer uma maneira mais familiar de manipular e limpar dados para pessoas novas em Python ou
# no pandas.
pysqldf = lambda q: sqldf(q, globals())
    #   """
    #   Função para calcular quais são os quatro estados que possuem maior frequência de relatos, e quais
    #   são os tipos de OVNIS mais populares.
    #   INPUT:
    #   q: É a o código da Query (uma query é um pedido de uma informação ou de um dado. Esse pedido também pode
    #   ser entendido como uma consulta, uma solicitação ou, ainda, uma requisição.)
    #   globals(): O conjunto de variáveis globais do ambiente, definidas pelo usuário.
    #   OUTPUT:
    #    DataFrame com o resultado da query traz quais são os quatro estados que possuem maior frequência de
    #   relatos, assim como os tipos de OVNIS mais populares.
    #   """
```

```
# cria um novo DataFrame a partir do código da query.
df_ovnis_4states=pysqldf("""  SELECT  STATE,COUNT(CASE  WHEN  SHAPE  ==
'Light' THEN 1 END) Light,COUNT(CASE WHEN SHAPE == 'Circle' THEN 1 END)
Circle,COUNT(CASE   WHEN   SHAPE   ==   'Fireball'   THEN   1   END)
Fireball,COUNT(CASE WHEN SHAPE == 'Sphere' THEN 1 END) Sphere
FROM df_ovnis WHERE
STATE == 'CA' or STATE == 'WA' or STATE == 'FL' OR STATE ==GROUP BY STATE
""" )

# salva o novo conjuntos de dados para o arquivo .csv
df_ovnis_4states.to_csv('quatro_estados_com_mfr_e_ovnis_mp.csv')

# plot do gráfico de barras agrupadas
df_ovnis_4states.plot.bar(x     =     'STATE',     y     =     ['Light',
'Circle','Fireball','Sphere'])

# plot do gráfico de barras empilhadas (a única coisa que muda nesse
código é que passamos o argumento stacked como verdadeiro,
# o que faz com que as barras fiquem empilhadas)
df_ovnis_4states.plot.bar(x     =     'STATE',     y     =     ['Light',
'Circle','Fireball','Sphere'],stacked=True)

# usa_states = pd.read_csv("df_OVNI_preparado.csv")
relatos = pd.read_csv("ovnis_clean.csv")


caCircle = relatos.loc[(relatos['STATE'] == 'CA') & (relatos['SHAPE'] ==
'Circle')]
waCircle = relatos.loc[(relatos['STATE'] == 'WA') & (relatos['SHAPE'] ==
'Circle')]
flCircle = relatos.loc[(relatos['STATE'] == 'FL') & (relatos['SHAPE'] ==
'Circle')]
txCircle = relatos.loc[(relatos['STATE'] == 'TX') & (relatos['SHAPE'] ==
'Circle')]

caFireball = relatos.loc[(relatos['STATE'] == 'CA') & (relatos['SHAPE']
== 'Fireball')]
waFireball = relatos.loc[(relatos['STATE'] == 'WA') & (relatos['SHAPE']
== 'Fireball')]
flFireball = relatos.loc[(relatos['STATE'] == 'FL') & (relatos['SHAPE']
== 'Fireball')]
```

```
txFireball = relatos.loc[(relatos['STATE'] == 'TX') & (relatos['SHAPE'] == 'Fireball')]

caLight = relatos.loc[(relatos['STATE'] == 'CA') & (relatos['SHAPE'] == 'Light')]
waLight = relatos.loc[(relatos['STATE'] == 'WA') & (relatos['SHAPE'] == 'Light')]
flLight = relatos.loc[(relatos['STATE'] == 'FL') & (relatos['SHAPE'] == 'Light')]
txLight = relatos.loc[(relatos['STATE'] == 'TX') & (relatos['SHAPE'] == 'Light')]

caSphere = relatos.loc[(relatos['STATE'] == 'CA') & (relatos['SHAPE'] == 'Sphere')]
waSphere = relatos.loc[(relatos['STATE'] == 'WA') & (relatos['SHAPE'] == 'Sphere')]
flSphere = relatos.loc[(relatos['STATE'] == 'FL') & (relatos['SHAPE'] == 'Sphere')]
txSphere = relatos.loc[(relatos['STATE'] == 'TX') & (relatos['SHAPE'] == 'Sphere')]


print(len(waCircle))

Circle = [len(caCircle), len(waCircle), len(flCircle), len(txCircle)]
Fireball    =    [len(caFireball),    len(waFireball),    len(flFireball), len(txFireball)]
Light = [len(caLight), len(waLight), len(flLight), len(txLight)]
Sphere = [len(caSphere), len(waSphere), len(flSphere), len(txSphere)]

barWidth = 0.2
plt.figure(figsize=(10,5))
r1 = np.arange(len(Circle))
r2 = [ x + barWidth for x in r1]
r3 = [ x + barWidth for x in r2]
r4 = [ x + barWidth for x in r3]

plt.bar(r1, Circle, color='#8F3988', width=barWidth, label='Circle')
plt.bar(r2, Fireball, color='#7C00F5', width=barWidth, label='Fireball')
plt.bar(r3, Light, color='#F6E43E', width=barWidth, label='Light')
plt.bar(r4, Sphere, color='#C29F13', width=barWidth, label='Sphere')

plt.xlabel('Estados')
```

```
plt.xticks([r + barWidth for r in range(len(Circle))], ['CA', 'FL', 'TX',
'WA'])
plt.ylabel('Shapes')

plt.legend()
plt.show()

#Pegando os dados da biblioteca zipcodes (somente os dados que nos serão
úteis), atribuindo ao dataframe "df_zipcodes"
zipcodes_json = zpc.list_all()
df_zipcodes = pd.DataFrame(zipcodes_json)

df_zipcodes = df_zipcodes[['zip_code','city','state','lat','long']]
df_zipcodes

#excluindo os dados duplicados
df_zipcodes.drop_duplicates(subset=['city','state'],inplace=True)
df_zipcodes
#salvando no arquivo "zip_code.csv"
df_zipcodes.to_csv('zip_code.csv', index=False)

#Lendo os dados do arquivo "ovnis_clean.csv", salvo na Sprint passada,
atribuindo seus valores ao dataframe "df_dados"
df_dados = pd.read_csv(".//ovnis_clean.csv")
df_dados

#alterando a fonte dos dados "city", "state", "lat" e "long" para que
fique igual ao do dataframe "df_zipcodes.csv"
df_dados.rename(columns={"CITY":"city", "STATE":"state"}, inplace=True)
df_dados

#Fazendo um merge entre os dataframes "df_zipcodes" e "df_dados" para
adiocionar a latitude e a longitude encontradas no dataframe "df_zipcodes"
nas cidades encontradas no dataframe "df_dados" e atribuindo esses dados
ao dataframe "df_todos_os_dados"
df_todos_os_dados = pd.merge(df_zipcodes,df_dados, on=["city","state"])
df_todos_os_dados

#selecionando os dados que iremos utilizar e salvando esse novo dado no
dataframe "df_todos_os_dados".
df_todos_os_dados = df_todos_os_dados[["city","state","lat","long"]]
```

```
df_todos_os_dados.to_csv("dados_para_mapa.csv")
```

```
#devido a um bug,é necessário ler o arquivo novamente, caso contrário o
panda dá erro e não consegue plotar o mapa.
df_todos_os_dados = pd.read_csv('./dados_para_mapa.csv')

#criando um array com a latitude e a longitude dos paises do datraframe
"df_todos_os_dados"
coordenadas = []
for                              lat,long                              in
zip(df_todos_os_dados.lat.values,df_todos_os_dados.long.values):
  coordenadas.append([lat,long])

#com o geopandas, buscaremos as coordenadas dos EUA para utilizar na faze
seguinte
gpds.tools.geocode("United        States",        provider="nominatim",
user_agent="Intro Geocode")

#Intancia um novo mapa com o folium, passando as coordenadas dos EUA
mapa = folium.Map(location=[-100.44588, 39.78373],zoom_start

#tratamento de erro de entradas, devido a quantidade da dados utilizado,
é necessário aumentar o limite de entradas para evitar erros.
sys.setrecursionlimit(10**6)

#Cria um mapa de calor passando como parâmetros as coordenadas das cidades
mapa.add_child(plugins.HeatMap(coordenadas))
mapa

#Separando os dados onde o estado seja "california" e atribuindo ao
dataframe "df_california"
df_california = df_todos_os_dados[df_todos_os_dados['state']=='CA']
df_california

#criando um array com a latitude e a longitude dos paises do datraframe
"df_california"
coordenadas_ca = []
for lat,long in zip(df_california.lat.values,df_california.long.values):
  if [df_california['state']=='CA']:
    coordenadas_ca.append([lat,long])
#com o geopandas, buscaremos as coordenadas da California para utilizar
na faze seguinte
gpds.tools.geocode("California", provider="nominatim", user_agent="Intro
Geocode")
```

```
#Intancia um novo mapa com o folium, passando as coordenadas da California
mapa_ca = folium.Map(location=[-118.75600, 36.70146],zoom_start=3)

#Cria um mapa de calor passando como parâmetros as coordenadas das cidades
da california
mapa_ca.add_child(plugins.HeatMap(coordenadas_ca))
mapa_ca
```

Link do código do script ‘Limpeza dos dados’:

https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20limpeza%20dos%20dados/Script_limpeza_dos_dados_5_7.ipynb

```
#Realizar os Imports e instalações das bibliotecas
import pandas as pd
import numpy as np

# Abra o arquivo CSV e mostre algumas linhas
df_ovnis = pd.read_csv('ovnis_tabela.csv')
df_ovnis.head()

# verifica as colunas que possuem valores vazios
df_ovnis.isnull().sum()

# elimina os valores vazios das colunas 'CITY', 'STATE' e 'SHAPE'.
df_ovnis.dropna(subset=['CITY','STATE','SHAPE'], inplace=True)

# verifica se os valores vazios foram deletados do DataFrame
df_ovnis.isnull().sum()

df_ovnis.info()

# verifica quantos valores existem para a coluna 'STATE'
df_ovnis['STATE'].unique()

# seleciona os estados que não são dos Estados Unidos
df_remove = df_ovnis.loc[(df_ovnis['STATE'] == 'NS') | (df_ovnis['STATE']
== 'QC') | (df_ovnis['STATE'] == 'ON') | (df_ovnis['STATE'] == 'BC') |
(df_ovnis['STATE']   ==   'AB')   |   (df_ovnis['STATE']   ==   'NT')   |
(df_ovnis['STATE']   ==   'NB')   |   (df_ovnis['STATE']   ==   'PE')   |
(df_ovnis['STATE']   ==   'SK')   |   (df_ovnis['STATE']   ==   'SA')   |
```

```
(df_ovnis['STATE'] == 'MB') | (df_ovnis['STATE'] == 'NF') |
(df_ovnis['STATE'] == 'YT')]
df_remove.head()
ultimo_df = df_ovnis.drop(df_remove.index)

# verifica se conseguiu realizar a limpeza
ultimo_df.info()

# segunda verificação
ultimo_df['STATE'].unique()

# salva os novos conjuntos de dados para a próxima seção
ultimo_df.to_csv('ovnis_clean.csv', index=False)

# Abra o arquivo CSV e mostre algumas linhas
df_ovnis = pd.read_csv('ovnis_clean.csv')
df_ovnis.head()
#verifica quantos valores existem para a coluna 'SHAPE'
df_ovnis['SHAPE'].unique()

# verifica quantos valores da coluna 'SHAPE' possuem uma ocorrência
superior a 1000.
df_ovnis['SHAPE'].value_counts() > 1000

# seleciona os formatos que possuem ocorrências inferiores ou iguais a
1000.
df_remove = df_ovnis.loc[df_ovnis['SHAPE'].str.contains('Cross') |
df_ovnis['SHAPE'].str.contains('Cone') |
df_ovnis['SHAPE'].str.contains('Egg') |
df_ovnis['SHAPE'].str.contains('Teardrop') |
df_ovnis['SHAPE'].str.contains('Chevron') |
df_ovnis['SHAPE'].str.contains('Diamond') |
df_ovnis['SHAPE'].str.contains('Cylinder')]
df_remove
# aqui eliminamos as ocorrências inferiores ou iguais a 1000 do DataFrame.
# desta forma, só sobram as ocorrências superiores a 1000.
ultimo_df = df_ovnis.drop(df_remove.index)

# aqui verificamos se os valores inferiores foram mesmo eliminados
ultimo_df['SHAPE'].value_counts() > 1000

# salva os novos conjuntos de dados para a próxima seção
ultimo_df.to_csv('df_OVNI_limpo.csv', index=False)
```

Link do script ‘Acréscimo de variáveis’

https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20acr%C3%A9scimo%20de%20vari%C3%A1veis/Script_acrescimo_de_variaveis_5_8.ipynb

```
#Realizar os Imports e instalações das bibliotecas
import pandas as pd
from datetime import datetime as dt
import numpy as np
import locale

# Esse código muda a linguagem do sistema para o português brasileiro
# no google colab não funciona
# import locale
# Isso muda o formato do datetime pra pt_BR.
#locale.setlocale(locale.LC_ALL, 'pt_BR.UTF-8')

#locale.setlocale(locale.LC_ALL, 'pt_BR')

# locale.getlocale()
# ('en_US', 'UTF-8')
# Isso muda o formato do datetime pra pt_BR.
# locale.setlocale(locale.LC_ALL, 'pt_BR')


# Abra o arquivo CSV e mostre algumas linhas
df_ovnis = pd.read_csv('df_OVNI_limpo.csv')
df_ovnis.head()
```

```
#referência:                          https://www.programiz.com/python-programming/datetime/strftime
# primeiro convertemos a coluna 'DATE/TIME' para datetime
df_ovnis['DATE/TIME'] = pd.to_datetime(df_ovnis['DATE/TIME'])
# aqui criamos a coluna 'SIGHT_DATE' que possui o dia, o mês e o ano
df_ovnis['SIGHT_DATE'] = df_ovnis['DATE/TIME'].dt.strftime('%m/%d/%-y')
# já a coluna 'SIGHT_TIME' possui a hora e os minutos.
df_ovnis['SIGHT_TIME'] = df_ovnis['DATE/TIME'].dt.strftime('%H:%M')
# nessa parte nós eliminamos a coluna 'DATE_TIME'
df_ovnis.drop(['DATE/TIME'], axis=1, inplace=True)
df_ovnis

# Inicialmente eu iria separar a coluna dessa forma, mas como estamos trabalhando
# com dados de tempo, achei melhor trabalhar com datetime,
#                df_ovnis[['SIGHT_DATE','SIGHT_TIME']]                    = df_ovnis['DATE/TIME'].str.split(" ", n=1, expand=True)
# df_ovnis.drop(['DATE/TIME'], axis=1, inplace=True)
# df_ovnis
```

```
# primeiro convertemos a coluna 'SIGHT_DATE' para datetime
df_ovnis['SIGHT_DATE'] = pd.to_datetime(df_ovnis['SIGHT_DATE'])
# aqui criamos a coluna 'SIGHT_WEEKDAY' que diz o dia da semana
df_ovnis['SIGHT_WEEKDAY'] = df_ovnis['SIGHT_DATE'].dt.strftime('%A')
df_ovnis['SIGHT_DATE'] = df_ovnis['SIGHT_DATE'].dt.strftime('%m/%d/%-y')
```

```
df_ovnis
```

```
df_ovnis['SIGHT_DATE'] = pd.to_datetime(df_ovnis['SIGHT_DATE'])
df_ovnis['SIGHT_DAY'] = df_ovnis['SIGHT_DATE'].dt.strftime('%-d')
df_ovnis['SIGHT_MONTH'] = df_ovnis['SIGHT_DATE'].dt.strftime('%-m')
df_ovnis['SIGHT_DATE'] = df_ovnis['SIGHT_DATE'].dt.strftime('%m/%d/%-y')
df_ovnis
```

```
# salva os novos conjuntos de dados para a próxima seção
df_ovnis.to_csv('df_OVNI_preparado.csv', index=False)
```

Link do script “Dados no MongoDB” e “Análise”: https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20dados%20no%20MongoDB/Script_dados_no_MongoDB_5_10.ipynb

```
#Realizar os Imports e instalações das bibliotecas
!curl ipecho.net/plain
!pip install pymongo
!pip install dnspython
import pandas as pd
import pymongo
from pymongo import MongoClient
import pprint

#abrindo conexão com Atlas:MongoDB - Password: VKmFe5ghPBIIi3K1
client =
pymongo.MongoClient("mongodb+srv://Elizeu:VKmFe5ghPBIIi3K1@teste.tprnx.
mongodb.net/<dbname>?retryWrites=true&w=majority")
#criando um Banco de dados chamado ovni
db = client.ovni
#Criando uma coleção chamada ovnis.
ovnis = db.ovnis
#Leitura do arquivo df_OVNI_preparado.csv para verificar se ele foi
importado
df_ovnis = pd.read_csv('df_OVNI_preparado.csv')
df_ovnis.head()

#Transformar CSV em um dictionary
#ovnis.insert_many(df_ovnis.to_dict('df_ovnis_dic'))
import csv
df_ovnis_dic = []

reader = csv.DictReader(open('df_OVNI_preparado.csv'))

for line in reader:
```

```
    df_ovnis_dic.append(line)

#pprint.pprint(df_ovnis_dic)

# Inserir na coleção criada todos os registros do df_OVNI_preparado
ovnis.insert_many(df_ovnis_dic)

#ANÁLISE

#Contar e mostrar quantos documentos há na coleção ovnis.
ovnis.count_documents({})

#Resgatar todos os documentos (registros) da coleção ovnis e ordenar
por tipo (shape)
for ovni in ovnis.find({}).sort("shape"):
  pprint.pprint(ovni)

#Resgatar todos os documentos (registros) da coleção ovnis e ordenar
por tipo (shape)
for ovni in ovnis.find({}).sort("shape"):
  pprint.pprint(ovni)

# Verificar quantas ocorrências existem por estado.
from bson.son import SON
# agregamos usando a função aggregate() de collections,
# e atribuimos o valor "1" = "True" para a operação $sum para contar
todas as ocorrências.
resultados = db.ovnis.aggregate([{
                    "$group": {
                        "Views": {"$sum": 1},
                        "_id": "$STATE"
                    }
}])

for resultado in resultados:
  print(resultado)
#for ovni in ovnis.find({}):
#  pprint.pprint(ovni)

#pprint.pprint(list(db.ovnis))

# Buscar todas as ocorrências da cidade Phoenix.
for ovni in ovnis.find({"CITY": "Phoenix"}).sort("shape"):
```

```
  pprint.pprint(ovni)
```

Link do script “Análise Temporal”: https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g2-efl/blob/master/Script%20An%C3%A1lise%20temporal/C%C3%B3pia_de_C%C3%B3pia_de_C%C3%B3pia_de_5_12_An%C3%A1lise_temporal.ipynb

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#Ordenar as observações de forma ascendente temporalmente (da
observação mais antiga para a observação mais recente).
df_ovnis = pd.read_csv("df_OVNI_preparado.csv", index_col=[0])

#Cria um novo dataframe com os dados da cidade de Phoenix
df_ovnis = df_ovnis[df_ovnis['CITY']=='Phoenix']

#agrupa por data
df_views = df_ovnis
df_views['Views'] =
df_views.groupby('SIGHT_DATE')['SIGHT_DATE'].transform('count')

#Excluir as colunas cujo os dados não são necessários
df_views = df_views.drop(columns=["SIGHT_MONTH"])
df_views = df_views.drop(columns=["SIGHT_DAY"])
df_views = df_views.drop(columns=["SIGHT_WEEKDAY"])
df_views = df_views.drop(columns=["SHAPE"])
df_views = df_views.drop(columns=["STATE"])

df_views = df_views.drop(columns=["CITY"])
df_views = df_views.drop(columns=["SIGHT_TIME"])
df_views.sort_values(by='SIGHT_DATE')

#Atribui o daframe a uma outro para manipulção mais livre
df_mes = df_ovnis
#Define o ano que quer ser visto no grafico
ano = 2017

#Transforma a coluna 'Sight_Date' em formato de data para separar o ano
```

```
df_mes['SIGHT_DATE'] = pd.to_datetime(df_mes['SIGHT_DATE'])
#Separa o ano em uma coluna propria
df_mes['SIGHT_YEAR'] = df_mes['SIGHT_DATE'].dt.strftime('%Y')
#transforma o ano em string
df_mes = df_mes[df_ovnis['SIGHT_YEAR'] == str(ano)]
#Conta quantas visualizacoes cada mes tem
df_mes["Views"] =
df_mes.groupby('SIGHT_MONTH')['SIGHT_MONTH'].transform('count')

#cria um vetor com todos os meses do ano
mes = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
#cria um vetor vazio que tera as views de cada mes
views = []

#no primeiro for ele vai rodar todos os 12 meses
for j in mes:
  #flag para nao duplicar o numero de views de meses repetidos
  flag = 0
  #segundo for que ira interar linha por linha do dataframe
  for i in df_mes.itertuples():
    if(df_mes.SIGHT_MONTH[i.Index] == j and flag == 0):
      #atribui o numero do mes correspondente (representado pelo j)
      views.append(df_mes.Views[i.Index])
      flag = 1

#preenche os meses cujo n tiveram observações registradas com o valor
zero
if(len(views) < 12):
  meses_sem_views = 12 - len(views)
  for i in range(meses_sem_views):
    views.append(0)

#----PLOTANDO O GRAFICO-----

#cria distancia entre as barras
x1 =  np.arange(len(views))

# Plota as barras
plt.bar(x1, views, width=0.5, label = 'Produto A')

# coloca o nome dos meses como label do eixo x
plt.xticks([x for x in range(len(views))], mes)
```

```
plt.title("Ano de "+ str(ano))
plt.xlabel('Month')
plt.ylabel('Views')
plt.show()

#Atribui o daframe a uma outro para manipulção mais livre#Atribui o
daframe a uma outro para manipulção mais livre
df_anos = df_ovnis

#Array vazio que tera os anos de 1997 a 2017
anos = []
#Array vazio que tera as views correspondennte ao periodo pedido
views = []
#Anos que aparecerao no grafico
anos_grafico = []

#Transforma a coluna 'Sight_Date' em formato de data para separar o ano
df_anos['SIGHT_DATE'] = pd.to_datetime(df_anos['SIGHT_DATE'])
#Separa o ano em uma coluna propria
df_anos['SIGHT_YEAR'] = df_anos['SIGHT_DATE'].dt.strftime('%Y')
#Conta quantas visualizacoes cada ano tem
df_anos["Views"] =
df_anos.groupby('SIGHT_YEAR')['SIGHT_YEAR'].transform('count')

#intera os anos de 1997 a 2017 no array 'anos'
for i in range(1997, 2018):
  anos.append(i)

#no primeiro for ele vai rodar todos os os anos que o array rececbeu
for j in anos:
  #flag para nao duplicar o numero de views de anos repetidos
  flag = 0
  #segundo for que ira interar linha por linha do dataframe
  for i in df_anos.itertuples():
    if(df_anos.SIGHT_YEAR[i.Index] == str(j) and flag == 0):
      #atribui o ano correspondente a view adicionada
      anos_grafico.append(str(j))
      #atribui o numero do ano correspondente (representado pelo j)
      views.append(df_anos.Views[i.Index])
      flag = 1

#Atribui o 'anos_grafico' como anos do eixo x e as views no eixo y
plt.plot(anos_grafico, views)
```

```
plt.xlabel('Year')
plt.ylabel('Views')
plt.title("Evolução de Observações da Cidade de Phoenix ao Longo dos
Anos de 1997 a 2017")
plt.xticks(rotation=45)

plt.show()

# Transformamos o 'Sight_date' de df_views em tipo Date.
df_views['SIGHT_DATE'] = pd.to_datetime(df_views['SIGHT_DATE'])
# Definimos o 'Sight_Date' como o index.
df_views.set_index('SIGHT_DATE', inplace = True)
df_views

# Isso separa os conjunto df_views no conjunto de treinamento
(df_train) e no conjunto de teste (df_test).
# O conjunto df_train possui 70% dos dados do conjunto df_views,
enquanto o df_test possui 30% dos dados.
df_train, df_test = df_views.iloc[:235, :], df_views.iloc[235:, :]

df_train.shape

df_train.head()

df_test.shape

df_test.head()

import numpy as np
from sklearn.model_selection import train_test_split
#df_ovnis_train = pd.read_csv("df_OVNI_preparado.csv")
x_train, x_test, y_train, y_test =
train_test_split(df_views.drop('Views',axis=1), df_views['Views'],
test_size=0.3, random_state=0, shuffle=False)

x_train.shape, y_train.shape

x_test.shape, y_test.shape

from statsmodels.tsa.statespace.sarimax import SARIMAX
% matplotlib inline
sarimax_model = SARIMAX(df_train, order=(1,0,0))
```

```
#sarimax_model = SARIMAX(df_ovnis['Views'], freq='MS', order=(1,1,1),
seasonal_order=(1,1,1,12),exog=df_ovnis['Views']).fit(x_train, y_train)
#sarimax_model = sm.tsa.statespace.SARIMAX(df_train, order=(1,0,0))
#res = sarimax_model(disp=False)
#res.summary()

res = sarimax_model.fit(disp=False)
res.summary()

df_forecast = res.forecast(101)

df_forecast

df_forecast = pd.DataFrame(res.forecast(101), columns=['Views'])

df_forecast

df_test

array_forecast = []
array_test = []
diferenca = []

for i in df_forecast.itertuples():
  array_forecast.append(float(df_forecast.Views[i.Index]))

for index, row in df_test.iterrows():
  teste = str(row["Views"])
  array_test.append(float(teste))

for i in range(101):
  diferenca.append(array_forecast[i] - array_test[i])

print(diferenca)


print("Erro Médio:")

# O Erro Médio é calculado quando calculamos o valor absoluto
# da diferença entre o forecast e df_teste, depois pegamos a media
desse vetor
# resultante.
np.mean(np.abs(diferenca))
```

```
print("Desvio Padrão:")
# O metódo que calcula o desvio padrão da amostra é .std()
# ddof modifica o divisor da sum dos quadrados das amostras-menos-
média.
# O divisor é N - ddof , onde o padrão ddof é 0 como você pode ver no
seu
# resultado.
np.std(diferenca, ddof=1)
```

3. Considerações Finais

O grupo conseguiu executar as atividades propostas com êxito, apesar das dificuldades.

4. Referências

MONGO DB. The mongoDB 4.4 Manual. Disponível em: https://docs.mongodb.com/manual/. Acesso em: 25/10/2020

SANTANA, Rodrigo. Plotando gráficos de um jeito fácil com Python. Disponível em: https://minerandodados.com.br/plotando-graficos-de-forma-facil-com-python/. Acesso em: 14/10/2020

GeeksforGeeks. Different ways to iterate over rows in Pandas Dataframe. Disponível em: https://www.geeksforgeeks.org/different-ways-to-iterate-over-rows-in-pandas-dataframe/. Acesso em: 23/10/2020

USP DCC IME. PANDA. Princípios de Algoritmos e Estruturas de Dados Usando Python. Disponível em: https://panda.ime.usp.br/algoritmos/static/algoritmos/10-matplotlib.html. Acesso em: 19/10/2020

Stick-Learn; Machine Learning in Python. Disponível em: https://scikit-learn.org/stable/. Acesso em: 10/10/2020