TECNOLOGIA EM SISTEMAS PARA INTERNET

RELATÓRIO DE PRÁTICA INTEGRADA DE CIÊNCIA DE DADOS E APRENDIZADO DE MÁQUINA PARA A SPRINT II

Davi Pereira Speck Alves
Marcus Vinicius da Silva Romero
Pedro Stewart Silva Borges
Vinícius dos Santos Evangelista Dias

Brasília - DF
25/06/2022

Sumário

Objetivos Gerais

- Explorar os dados a partir de gráficos e mapas;
- Delimitar o escopo de dados da base e otimizar tratamento;
- Transferir dados ao MongoDB e realizar consultas.

Descrição do problema

Ver os dados de maneira crua é um problema, pois são muitos e a visualização torna-se difícil. Por essa razão, faz-se necessária a criação de dados. Além disso, é preciso filtrar os dados, eliminando campos desnecessários e campos vazios, para em seguida colocá-los em uma base de dados, onde se torna mais fácil a consulta dos mesmos.

Desenvolvimento

O projeto será desenvolvido principalmente em Python, devido à alta gama de possibilidades que a mesma oferece para trabalhar com a manipulação de dados. Porém, mediante necessidade, certos pontos podem ser feitos a partir de outras linguagens também.

Tecnologias Utilizadas

- **Python**: Principal tecnologia do projeto, utilizada para manipulação de dados e *web scraping*, com juntamente com as bibliotecas “Requests”, “Beautiful Soup”, “Pandas”, “Pandasql”, “Matplotlib”, “Altair”, “Folium” e “Geopandas”;
- **Javascript**: Utilizado para filtrar por “mês/ano” uma coluna de um arquivo .csv. Para trabalhar com CSV no Javascript, foram utilizadas as seguintes bibliotecas: csv-parser e objects-to-csv, além da biblioteca nativa da linguagem para trabalhar com arquivos. Para montar os gráficos em Javascript, usamos a biblioteca ChartJS. A filtragem, contudo, foi feita manualmente sem uso de bibliotecas. Mais adiante, para o armazenamento dos dados em uma base MongoDB foi utilizada a biblioteca Mongoose.

Link do repositório no GitHub

https://github.com/infocbra/pratica-integrada-cd-e-am-2022-1-dmpv.git

Sprint II

1 - Exploração com gráficos e mapas

O código abaixo utiliza duas funções de distinção. Essas funções trazem como resposta quais são os elementos que compõem determinado conjunto, sem repeti-los. Isso é

útil para fazer contagens, como aquelas que são feitas pela função “formasEstado” o que retorna quantas formas (shapes) existem em cada um dos estados. Veja abaixo um exemplo de uma função de distinção.

```
      function distingueEstados(lista) {
        let estadosDistinguidos = [];
        for (let i = 0; i < lista.length; i++) {
          if (!thereIs(estadosDistinguidos, lista[i].State)) {
            estadosDistinguidos.push(lista[i].State);
          }
        }

        return estadosDistinguidos;
      }
```

Em seguida, limitamos o objeto gerado pela função “formasEstado” apenas para os estados com mais ocorrências, os quais já sabíamos de antemão por meio de consultas prévias em Python. O objeto é exibido na tela do terminal, de modo que se torna possível copiá-lo manualmente, já que se trata de pouca informação.

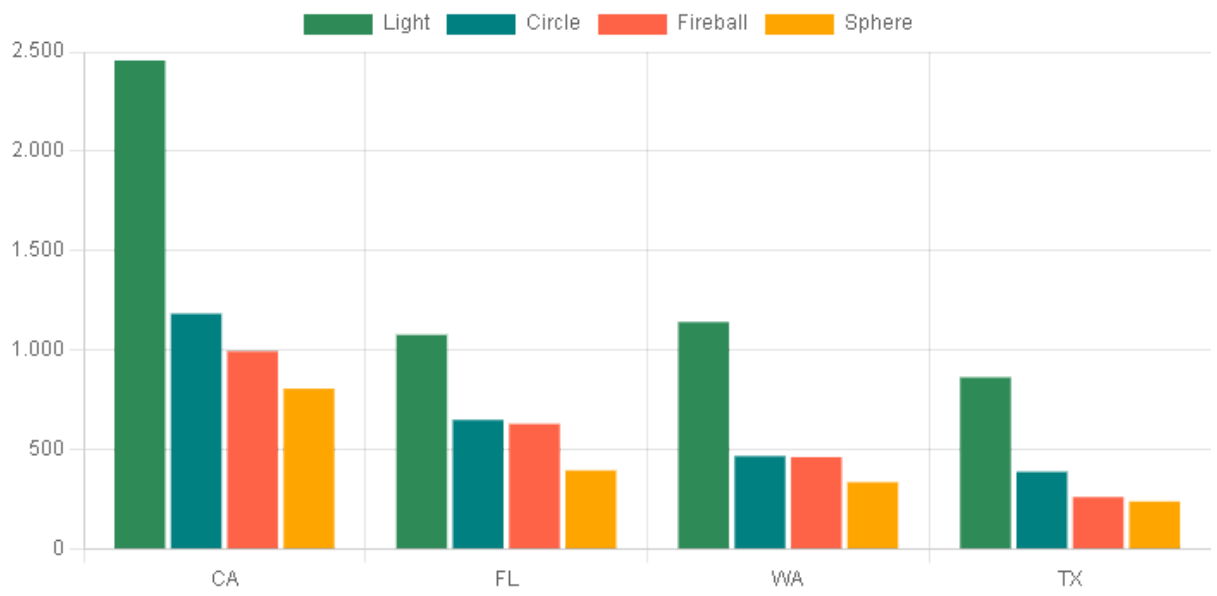
Embora todos os shapes sejam exibidos, eles aparecem de maneira ordenada, então basta pegar os primeiros que já se sabe aqueles que ocorreram mais vezes no estado em questão.

```
let resultado = [];

fs.createReadStream('todos_ovnis.csv')
.pipe(csvParser())
.on('data', (data) => resultado.push(data))
.on('end', () => {
  let estadosDistinguidos = distingueEstados(resultado);
  let formasDistinguidas = distingueFormas(resultado);
  let resultadoFormasEstados = formasEstados(resultado,
estadosDistinguidos, formasDistinguidas);

  let estadosMaioresQuantidades = [];
  for (let i = 0; i < resultadoFormasEstados.length; i++) {
    switch(resultadoFormasEstados[i].estado) {
      case 'CA':
      case 'WA':
      case 'FL':
      case 'TX':
        estadosMaioresQuantidades.push(resultadoFormasEstados[i]);
        break;
    }
  }

  console.log(estadosMaioresQuantidades);
});
```

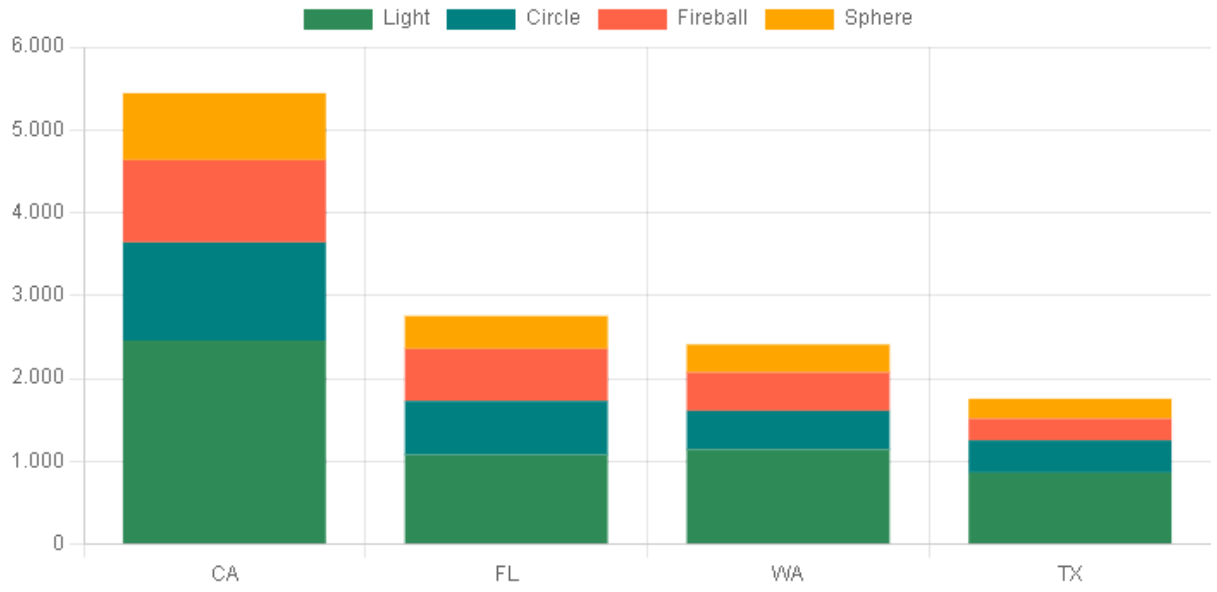
O gráfico então é criado dentro da tag HTML <script> com a biblioteca ChartJS, a partir das informações do objeto exibido no terminal pelo código acima. Os valores em “data” correspondem exatamente ao objeto que foi obtido manualmente do terminal.

```
// Dados que serão exibidos nos gráficos
let data = {
  labels: ["CA", "FL", "WA", "TX"],
  datasets: [{
    label: "Light",
    backgroundColor: "seagreen",
    data: [2457, 1078, 1142, 864]
  }, {
    label: "Circle",
    backgroundColor: "teal",
    data: [1184, 650, 468, 389]
  }, {
    label: "Fireball",
    backgroundColor: "tomato",
    data: [995, 630, 463, 262]
  }, {
    label: "Sphere",
    backgroundColor: "orange",
    data: [807, 396, 337, 240]
  }]
};

// Primeiro gráfico de barras lado a lado
let grafico1 = new Chart(g1, {
  type: 'bar',
  data: data,
  options: {
    barValueSpacing: 20
  }
});

// Segundo gráfico de barras empilhadas
let grafico2 = new Chart(g2, {
  type: 'bar',
  data: data,
  options: {
    barValueSpacing: 20,
    scales: {
      x: {
        stacked: true,
      },
      y: {
        stacked: true
      }
    }
  }
});
```

1.1 - Gráfico de barras agrupadas



1.2 - Gráfico de barras empilhadas



1.3 - Mapa do país inteiro (EUA)

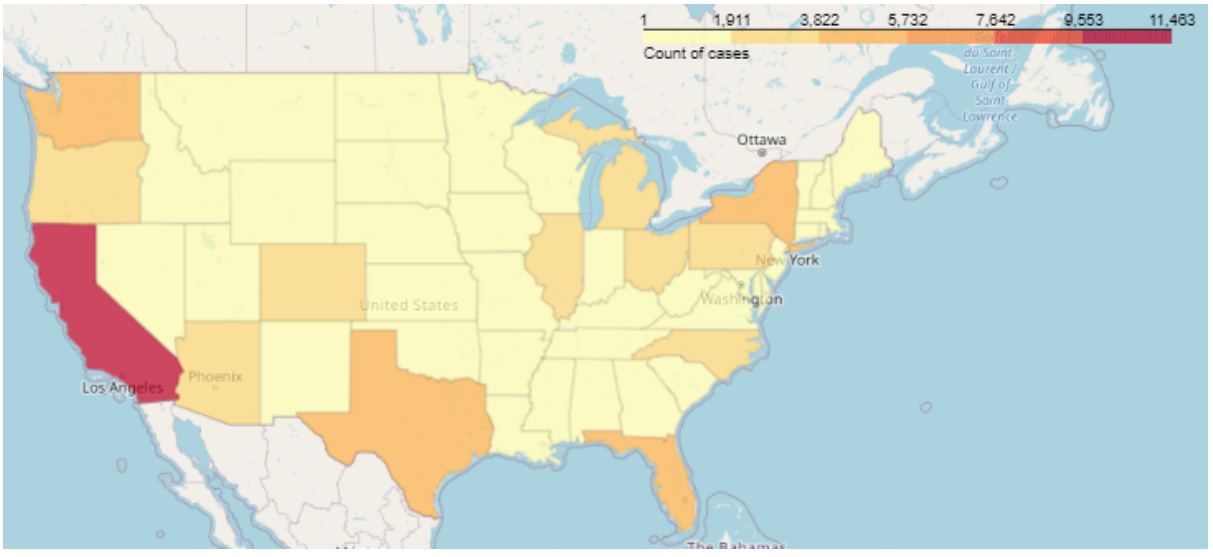
1.3.1 - Iniciando "folium.Map" a partir de Montana com o intuito de manter os EUA centralizado

```
m = folium.Map(location=[40, -95], zoom_start=4)
```

1.3.2 - Montando um gráfico de calor baseado nos locais dos EUA com maior quantidade de ocorrências a partir da biblioteca "Folium"

```
url = (

"https://raw.githubusercontent.com/python-visualization/folium/master/examples/data"
)
state_geo = f"{url}/us-states.json"
state_unemployment = f"{url}/US_Unemployment_Oct2012.csv"
state_data = pd.read_csv(state_unemployment)

cities = pd.read_csv('todos_ovnis.csv')

q = """
SELECT State, COUNT(State) as Count, Country FROM cities WHERE Country = 'USA' GROUP BY State HAVING COUNT(State) > 0 ORDER BY Count(State);
"""

df = pandasql.sqldf(q, locals())
merge = pd.merge(df, state_data, how = 'inner')
df_merge = df.drop_duplicates(subset='State')
```

Saída:



1.4 - Mapa do Estado da Califórnia (EUA)

1.4.1 - Criando um dataframe com todas as cidades da Califórnia, contendo, respectivamente, latitude, longitude e quantidade de ocorrências nas mesmas.

```
url = (

"https://raw.githubusercontent.com/python-visualization/folium/master/e
xamples/data"
)
county_geo = f"{url}/us_counties_20m_topo.json"
state_unemployment = f"{url}/US_Unemployment_Oct2012.csv"
state_data = pd.read_csv(state_unemployment)

uscities = pd.read_csv('uscities.csv')

uscities_new = uscities.drop(['city_ascii', 'state_name',
'county_fips', 'county_name', 'population', 'density', 'source',
'military', 'incorporated', 'timezone', 'ranking', 'zips', 'id'],
axis=1)

q = """
SELECT City, lat, lng FROM uscities_new WHERE state_id = 'CA' ORDER BY
City;
"""

df_cities = pandasql.sqldf(q, locals())

df_cities_renamed = df_cities.rename(columns={'city': 'City'})

uscities = pd.read_csv('todos_ovnis.csv')

q = """
SELECT State, City, COUNT(City) as Count FROM uscities WHERE State =
'CA' GROUP BY City HAVING COUNT(City) > 0 ORDER BY COUNT(City) DESC;
"""

df = pandasql.sqldf(q, locals())
cities_merge = pd.merge(df_cities_renamed, df, how = 'inner')
```

1.4.2 - Definindo um array com latitudes e longitudes mínimas e máximas com o objetivo de definir imagem usada para plotar a contagem de ocorrências.

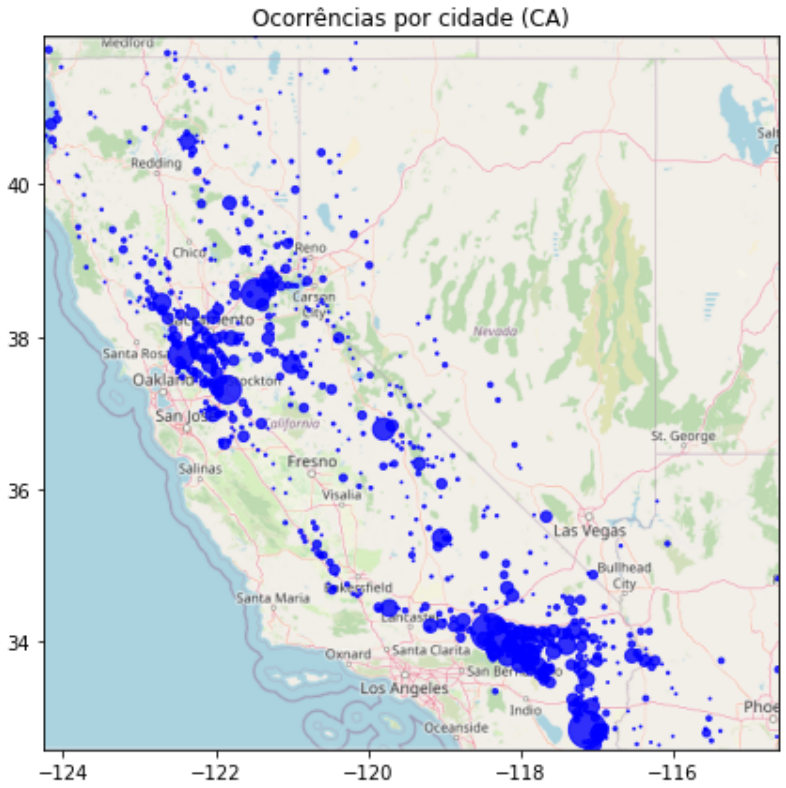
```
BBox = [cities_merge['lng'].min(), cities_merge['lng'].max(),
cities_merge['lat'].min(), cities_merge['lat'].max()]
```

1.4.3 - Importando uma imagem montada com o bloco de código passado.

```
ruh_m = pltimread('map.png')
```

1.4.4 - Mostrando contagem plotada em uma imagem da Califórnia.

```
fig, ax = plt.subplots(figsize = (8,7))

ax.scatter(cities_merge.lng,cities_merge.lat, zorder=0.5, alpha= 0.8,
c='b', s=cities_merge.Count)

ax.set_title('Ocorrências por cidade (CA)')
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])

ax.imshow(ruh_m, zorder=0, extent = BBox, aspect= 'equal')
```



1.5 - (Questão) Onde na Califórnia está localizada a maior quantidade de visualizações de objetos voadores não identificados? Investigue e descreva a possível razão para esse local ter a maior quantidade de visualizações.

Resposta: San Diego com 392 ocorrências. Essa quantidade pode se justificar através de sua própria localização, no estado mais populoso dos Estados Unidos, e ser também a segunda cidade mais populosa da região. Além disso, é dificultada a visualização de estrelas no céu devido à poluição luminosa, portanto, luzes inusitadas são dadas como objetos voadores não identificados, informações essas constatadas pelos dados.

2 - Preparação dos dados

2.1 - Script da etapa

1. Carregar o seu arquivo OVNIS.csv em um dataframe;
2. Remover registros que tenham valores vazios (None, Unknown, ...) para City, State e Shape;
3. Manter somente os registros referentes aos 51 estados dos Estados Unidos (Links para um site externo.);
4. Remover variáveis irrelevantes para a análise (Duration, Summary e Posted);
5. Manter somente os registros de Shapes mais populares (com mais de 1000 ocorrências);
6. Salvar o dataframe final em um arquivo CSV com o nome "df_OVNI_limpo".

Porém, esta etapa foi realizada em JavaScript. A função “filtraDadosOvnis” realiza todo o trabalho de filtragem.

```
const fs = require('fs');
const csvParser = require('csv-parser');
const ObjectsToCsv = require('objects-to-csv')


function filtraDadosOvnis(lista, resultadoOcorrenciasTodasFormas) {
  let resultado = [];
  for (let i = 0; i < lista.length; i++) {

    delete lista[i].Duration; delete lista[i].Summary; delete lista[i].Posted;
delete lista[i].Images;

    let formaInvalida = false;
    for (let j = 0; j < resultadoOcorrenciasTodasFormas.length; j++) {
      if (lista[i].Shape === resultadoOcorrenciasTodasFormas[j].forma &&
resultadoOcorrenciasTodasFormas[j].ocorrencias <= 1000) {
        formaInvalida = true;
      }
    }

    if (!formaInvalida && lista[i].Country === 'USA') {
      if (!(lista[i].City === 'None' || lista[i].City === 'Unknown' ||
lista[i].City === '')) {
        if (!(lista[i].State === 'None' || lista[i].State === 'Unknown' ||
lista[i].State === '')) {
          if (!(lista[i].Shape === 'None' || lista[i].Shape === 'Unknown' ||
lista[i].Shape === '')) {
            resultado.push(lista[i]);
          }
        }
      }
    }
  }

  return resultado;
}

let resultado = [];
fs.createReadStream('TODOS_OVNIS.csv')
.pipe(csvParser())
.on('data', (data) => resultado.push(data))
```

```
.on('end', () => {
  let formasDistinguidas = distingueFormas(resultado);
  let resultadoOcorrenciasTodasFormas =
ocorrenciasTodasFormas(formasDistinguidas, resultado);

  resultadoOcorrenciasTodasFormas.sort((a, b) => {
      return a.ocorrencias - b.ocorrencias;
  }).reverse();

  toCsv(filtraDadosOvnis(resultado, resultadoOcorrenciasTodasFormas),
'OVNI_limpo');
});
```

2.2 - Script das novas colunas

1. Dividir o conteúdo da coluna Date / Time em duas novas colunas no mesmo dataframe e deletar a coluna Date / Time;
2. Fazer o mesmo procedimento para dias da semana;
3. Separar as variáveis mês (Month) e dia (Day);
4. Salvar o dataframe resultante em um arquivo .csv com o nome: 'df_OVNI_preparado'.

As etapas deste script foram realizadas no momento em que os dados foram armazenados no MongoDB. Portanto, na etapa seguinte.

3 - Dados no MongoDB

3.1 - Primeiro script da etapa

1. Criar um banco de dados chamado ovni;
2. Criar uma coleção com o nome ovnis;
3. Inserir na coleção criada todos os registros do df_OVNI_preparado.

Esta etapa foi feita em JavaScript, logo, a criação do banco de dados é feita automaticamente, pela biblioteca Mongoose. A coleção, por sua vez, é criada a partir do momento em que se tem um schema Mongoose.

3.2 - Schema Mongoose

```
const mongoose = require('mongoose');
const RegistroSchema = new mongoose.Schema({
  dateTime: {
    type: String,
    required: true,
```

```
  },
  city: {
    type: String,
    required: true,
  },
  state: {
    type: String,
    required: true,
  },
  country: {
    type: String,
    required: true,
  },
  shape: {
    type: String,
    required: true,
  }
});

const Registro = mongoose.model('Registro', RegistroSchema);
module.exports = Registro;
```

3.3 - Armazenamento no MongoDB (Atlas)

```
const fs = require('fs');
const mongoose = require('mongoose');
const csvParser = require('csv-parser');
const Registro = require('./models/Registro');

// conexão com o banco de dados (aOnd3yVydERDHZLu)
const dbURICloud =
"mongodb+srv://IFBOvnisPraticaIntegrada:aOnd3yVydERDHZLu@ovnis.rw2p3.mo
ngodb.net/?retryWrites=true&w=majority";
const dbURILocal = "mongodb://localhost/registroOvnis";

mongoose.connect(dbURICloud, { useNewUrlParser: true,
useUnifiedTopology: true })
  .then((result) => { console.log('Conexão estabelecida com o banco de
dados.'); })
  .catch((err) => console.log(err));

let resultado = [];
```

```
fs.createReadStream('OVNI_limpo.csv')
.pipe(csvParser())
.on('data', (data) => resultado.push(data))
.on('end', async () => {

  for (let i = 0; i < resultado.length; i++) {
    let data = resultado[i]['Date / Time'].split(' ')[0];
    let dia = data.split('/')[1];
    let mes = data.split('/')[0];
    let ano = data.split('/')[2];
    let diaSemana = getDiaSemana(parseInt(dia), parseInt(mes),
parseInt(ano));
    let hora = resultado[i]['Date / Time'].split(' ')[1] || '';

    const registro = await Registro.create({ date: data, day:
parseInt(dia), month: parseInt(mes), year: parseInt(ano), weekday:
parseInt(diaSemana), time: hora, city: resultado[i]['City'], state:
resultado[i]['State'], country: resultado[i]['Country'], shape:
resultado[i]['Shape'] });
  }

});
```

3.4 - Buscas no MongoDB (Atlas)

3.4.1 - Script das buscas

1. Contar e mostrar quantos documentos há na coleção ovnis;
2. Resgatar todos os documentos (registros) da coleção ovnis e ordenar por tipo (shape);
3. Verificar quantas ocorrências existem por estado;
4. Buscar todas as ocorrências da cidade Phoenix;
5. Buscar as ocorrências do estado da Califórnia e ocultar o id de cada documento (registro).

3.4.2 - Código em JavaScript para as buscas

```
async function buscasMongodb() {
  let busca = await Registro.find().count();
  console.log(busca);
  busca = await Registro.find().sort({ shape: 1});
  console.log(busca);
  busca = await Registro.aggregate([{$group: {_id:"$state", count:
{$sum: 1}}}])
```

```
  console.log(busca);
  busca = await Registro.find({city: "Phoenix"})
  console.log(busca);
  busca = await Registro.find({state: 'CA'}, {_id: 0})
  console.log(busca);
}

buscasMongodb();
```

3.4.3 - Resultados das buscas no MongoDB

3.4.3.1 - Contar e mostrar quantos documentos há na coleção ovnis.

```
let busca = await Registro.find().count();
console.log(busca);
```

Resultado: 78362

3.4.3.2 - Resgatar todos os documentos (registros) da coleção ovnis e ordenar por tipo (shape).

```
busca = await Registro.find().sort({ shape: 1});
console.log(busca);
```

Resultado:

```
[
  {
    _id: new ObjectId("62ccb4de5b4abe34bd2ce16f"),
    date: '8/26/17',
    day: 26,
    month: 8,
    year: 17,
    weekday: 6,
    time: '00:00',
    city: 'Arnold',
    state: 'PA',
    country: 'USA',
    shape: 'Changing',
    __v: 0
  },
  {
    _id: new ObjectId("62ccb4de5b4abe34bd2ce1a3"),
    date: '8/24/17',
```

```
    day: 24,
    month: 8,
    year: 17,
    weekday: 4,
    time: '05:00',
    city: 'Denver/Glendale',
    state: 'CO',
    country: 'USA',
    shape: 'Changing',
    __v: 0
  },
  {
    _id: new ObjectId("62ccb4de5b4abe34bd2ce1b1"),
    date: '8/23/17',
    day: 23,
    month: 8,
    year: 17,
    weekday: 3,
    time: '21:45',
    city: 'Treasure Island',
    state: 'FL',
    country: 'USA',
    shape: 'Changing',
    __v: 0
  },

etc. etc. etc.
```

3.4.3.3 - Verificar quantas ocorrências existem por estado.

```
busca = await Registro.aggregate([{$group: {_id:"$state", count: {$sum:
1}}}])
console.log(busca);
```

Resultado:

```
[
  { _id: 'MT', count: 570 },
  { _id: 'NC', count: 2238 },
  { _id: 'MS', count: 449 },
  { _id: 'WA', count: 4282 },
```

```
{ _id: 'NM', count: 916 },
{ _id: 'AL', count: 786 },
{ _id: 'AK', count: 402 },
{ _id: 'WI', count: 1498 },
{ _id: 'KS', count: 684 },
{ _id: 'Washington, DC', count: 1 },
{ _id: 'MN', count: 1253 },
{ _id: 'AR', count: 669 },
{ _id: 'FL', count: 4979 },
{ _id: 'SD', count: 214 },
{ _id: 'NY', count: 3441 },
{ _id: 'PA', count: 2884 },
{ _id: 'Ontario', count: 1 },
{ _id: 'CT', count: 1041 },
{ _id: 'MO', count: 1674 },
{ _id: 'CO', count: 1904 },
{ _id: 'TX', count: 3650 },
{ _id: 'WY', count: 231 },
{ _id: 'BC', count: 1 },
{ _id: 'NJ', count: 1671 },
{ _id: 'OH', count: 2644 },
{ _id: 'ID', count: 721 },
{ _id: 'ON', count: 2 },
{ _id: 'KY', count: 965 },
{ _id: 'DC', count: 107 },
{ _id: 'AB', count: 1 },
{ _id: 'SC', count: 1412 },
{ _id: 'VA', count: 1596 },
{ _id: 'MI', count: 2195 },
{ _id: 'RI', count: 361 },
{ _id: 'LA', count: 675 },
{ _id: 'GA', count: 1598 },
{ _id: 'IA', count: 760 },
{ _id: 'VT', count: 368 },
{ _id: 'HI', count: 404 },
{ _id: 'ND', count: 141 },
{ _id: 'CA', count: 10221 },
{ _id: 'IN', count: 1503 },
{ _id: 'ME', count: 687 },
{ _id: 'OR', count: 2137 },
{ _id: 'NH', count: 661 },
{ _id: 'NE', count: 409 },
{ _id: 'NV', count: 973 },
```

```
  { _id: 'OK', count: 826 },
  { _id: 'MD', count: 1104 },
  { _id: 'AZ', count: 3077 },
  { _id: 'MA', count: 1552 },
  { _id: 'DE', count: 244 },
  { _id: 'TN', count: 1381 },
  { _id: 'IL', count: 2784 },
  { _id: 'UT', count: 878 },
  { _id: 'NS', count: 1 },
  { _id: 'WV', count: 535 }
]
```

3.4.3.4 - Buscar todas as ocorrências da cidade Phoenix.

```
busca = await Registro.find({city: "Phoenix"})
console.log(busca);
```

Resultado:

```
[
  {
    _id: new ObjectId("62ccb4df5b4abe34bd2ce283"),
    date: '8/14/17',
    day: 14,
    month: 8,
    year: 17,
    weekday: 1,
    time: '00:20',
    city: 'Phoenix',
    state: 'AZ',
    country: 'USA',
    shape: 'Light',
    __v: 0
  },
  {
    _id: new ObjectId("62ccb4df5b4abe34bd2ce375"),
    date: '8/4/17',
    day: 4,
    month: 8,
    year: 17,
    weekday: 5,
    time: '21:15',
    city: 'Phoenix',
    state: 'AZ',
```

```
    country: 'USA',
    shape: 'Flash',
    __v: 0
  },
  {
    _id: new ObjectId("62ccb4df5b4abe34bd2ce459"),
    date: '7/26/17',
    day: 26,
    month: 7,
    year: 17,
    weekday: 3,
    time: '04:20',
    city: 'Phoenix',
    state: 'AZ',
    country: 'USA',
    shape: 'Fireball',
    __v: 0
  },
  {
    _id: new ObjectId("62ccb4e05b4abe34bd2ce63f"),
    date: '7/6/17',
    day: 6,
    month: 7,
    year: 17,
    weekday: 4,
    time: '21:25',
    city: 'Phoenix',
    state: 'AZ',
    country: 'USA',
    shape: 'Oval',
    __v: 0
  },

etc. etc. etc.
```

3.4.3.5 - Buscar as ocorrências do estado da Califórnia e ocultar o id de cada documento (registro).

```
busca = await Registro.find({state: 'CA'}, {_id: 0})
console.log(busca);
```

Resultado:

```
[
  {
```

```
  date: '8/31/17',
  day: 31,
  month: 8,
  year: 17,
  weekday: 4,
  time: '21:00',
  city: 'San Diego',
  state: 'CA',
  country: 'USA',
  shape: 'Rectangle',
  __v: 0
},
{
  date: '8/31/17',
  day: 31,
  month: 8,
  year: 17,
  weekday: 4,
  time: '20:15',
  city: 'E. Rio Vista',
  state: 'CA',
  country: 'USA',
  shape: 'Light',
  __v: 0
},
{
  date: '8/31/17',
  day: 31,
  month: 8,
  year: 17,
  weekday: 4,
  time: '10:00',
  city: 'Grass Valley',
  state: 'CA',
  country: 'USA',
  shape: 'Circle',
  __v: 0
},
{
  date: '8/30/17',
  day: 30,
  month: 8,
  year: 17,
```

```
    weekday: 3,
    time: '22:00',
    city: 'San Anselmo',
    state: 'CA',
    country: 'USA',
    shape: 'Sphere',
    __v: 0
  },
  {
    date: '8/30/17',
    day: 30,
    month: 8,
    year: 17,
    weekday: 3,
    time: '15:00',
    city: 'Vacaville or Fairfield (?) (near)',
    state: 'CA',
    country: 'USA',
    shape: 'Cigar',
    __v: 0
  },
  {
    date: '8/30/17',
    day: 30,
    month: 8,
    year: 17,
    weekday: 3,
    time: '04:30',
    city: 'Kearny Mesa',
    state: 'CA',
    country: 'USA',
    shape: 'Light',
    __v: 0
  },

etc. etc. etc.
```

Considerações finais

Nesta Sprint foram criados os primeiros gráficos que permitiram uma melhor compreensão sobre os avistamentos de ovnis.. Além disso, os dados foram filtrados, eliminando colunas inúteis e campos vazios, para em seguida armazená-los online no MongoDB por meio da ferramenta Atlas. Por fim, notou-se como é possível realizar consultas diversas com o MongoDB.

Referências

1. PyData Organization. **pandas documentation**. “pydata.org”, 2022. Disponível em: https://pandas.pydata.org/docs/. Acesso em: 26/06/2022
2. Reitz, Kenneth. **Requests HTTP For Humans**. Requests, 2022. Disponível em: https://requests.readthedocs.io/en/latest/. Acesso em: 26/06/2022
3. Richardson, Leonard. **Beautiful Soup Documentation**. Crummy, 2022. Disponível em: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. Acesso em: 26/06/2022
4. Openbase, Inc. **Beautiful Soup Documentation**. Openbase, 2022. Disponível em: https://openbase.com/python/pandasql. Acesso em: 26/06/2022
5. Story, Rob. **Folium**. “python-visualization”, 2013. Disponível em: https://python-visualization.github.io/folium/. Acesso em: 08/07/2022
6. Mafintosh. **CSV Parser**. Disponível em: <https://github.com/mafintosh/csv-parser>. Acesso em: 02/08/2022
7. ChartJS. **ChartJS**. Disponível em: <https://www.chartjs.org/>. Acesso em: 02/08/2022.
8. Mongoose. **Mongoose**. Disponível em: <https://mongoosejs.com>. Acesso em:02/08/2022.
9. Antonbot. **Objects To CSV**. Disponível em: <https://github.com/anton-bot/objects-to-csv>. Acesso em: 02/08/2022
10. GeoPandas. **GeoPandas**. Disponível em: <https://geopandas.org>. Acesso em: 02/08/2022
11. Matplotlib. **Matplotlib**. Disponível em: <https://matplotlib.org>. Acesso em: 02/08/2022
12. Python Visualization. **Folium**. Disponível em: <http://python-visualization.github.io/folium/>. Acesso em: 02/08/2022
13. Altair Viz. **Altair**. Disponível em: <https://altair-viz.github.io/>. Acesso em: 02/08/2022