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

- Visualizar a realidade dos dados ao longo do tempo, para uma cidade;
- Delimitar o conjunto de treinamento e de teste;
- Utilizar o modelo SARIMAX para realizar previsões.

Descrição do problema

Uma vez que a ideia é realizar previsões de cunho temporal, é preciso antes compreender de que maneira os dados se manifestam ao longo do tempo, em determinada região de interesse de estudo. Uma vez que isso é feito, determina-se um conjunto de treinamento e um de teste, a fim de que o modelo escolhido aprenda o padrão e prove que esse padrão está de acordo com novos dados que serão fornecidos. Para tal, será utilizado o modelo SARIMAX, que pressupõe a existência de uma sazonalidade.

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*, 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 III

1 - Recuperação e ordenação para Phoenix

1. Recuperar os dados (do MongoDB a partir do arquivo df_OVNI_preparado) de visualização sobre a cidade de Phoenix agrupados por dia, por mês e por ano;
2. Ordenar as observações de forma ascendente temporalmente (da observação mais antiga para a observação mais recente).

1.1 - Código em JavaScript

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

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

let resultado = [];

async function buscasMongodb() {
  let busca = await Registro.aggregate([
                        { $match: {city: "Phoenix"}},
                        { $group: {_id:"$date", count: {$sum: 1}} },
                        { $sort : {_id : 1} }
                    ]);
  console.log(busca);

  toCsv(busca, 'resultado-busca-phoenix');

}

buscasMongodb();
```

1.2 - Saída

```
Conexão estabelecida com o banco de dados.
[
  { _id: '1998-11-19', count: 1 },
  { _id: '1998-11-30', count: 1 },
  { _id: '1998-12-19', count: 1 },
  { _id: '1998-12-22', count: 1 },
  { _id: '1998-12-6', count: 1 },
  { _id: '1998-12-9', count: 1 },
  { _id: '1998-2-25', count: 1 },
  { _id: '1998-5-29', count: 3 },
  { _id: '1998-6-7', count: 1 },
```

```
{ _id: '1998-9-26', count: 1 },
{ _id: '1999-10-12', count: 1 },
{ _id: '1999-10-15', count: 1 },
{ _id: '1999-10-28', count: 1 },
{ _id: '1999-11-29', count: 1 },
{ _id: '1999-11-9', count: 1 },
{ _id: '1999-2-15', count: 1 },
{ _id: '1999-3-4', count: 1 },
{ _id: '1999-5-5', count: 1 },
{ _id: '1999-5-7', count: 1 },
{ _id: '1999-6-12', count: 1 },
{ _id: '1999-6-15', count: 1 },
{ _id: '1999-6-29', count: 1 },
{ _id: '1999-8-17', count: 1 },
{ _id: '1999-9-17', count: 1 },
{ _id: '2000-1-13', count: 1 },
{ _id: '2000-10-15', count: 1 },
{ _id: '2000-10-27', count: 1 },
{ _id: '2000-10-5', count: 1 },
{ _id: '2000-11-20', count: 1 },
{ _id: '2000-11-28', count: 1 },
{ _id: '2000-12-3', count: 1 },
{ _id: '2000-3-30', count: 1 },
{ _id: '2000-4-15', count: 1 },
{ _id: '2000-5-19', count: 1 },
{ _id: '2000-5-25', count: 1 },
{ _id: '2000-7-1', count: 1 },
{ _id: '2000-8-15', count: 1 },
{ _id: '2000-8-5', count: 1 },
{ _id: '2000-9-20', count: 1 },
{ _id: '2000-9-30', count: 1 },
{ _id: '2001-10-10', count: 1 },
{ _id: '2001-10-12', count: 1 },
{ _id: '2001-10-19', count: 1 },
{ _id: '2001-10-25', count: 2 },
{ _id: '2001-11-12', count: 1 },
{ _id: '2001-11-16', count: 1 },
{ _id: '2001-2-11', count: 1 },
{ _id: '2001-2-4', count: 1 },
{ _id: '2001-2-6', count: 1 },
{ _id: '2001-3-25', count: 1 },
{ _id: '2001-3-27', count: 1 },
{ _id: '2001-3-30', count: 1 },
{ _id: '2001-4-23', count: 1 },
{ _id: '2001-5-1', count: 1 },
{ _id: '2001-6-15', count: 1 },
{ _id: '2001-7-21', count: 1 },
{ _id: '2001-7-25', count: 1 },
{ _id: '2001-8-26', count: 1 },
{ _id: '2001-9-18', count: 1 },
{ _id: '2001-9-30', count: 1 },
{ _id: '2002-1-12', count: 1 },
{ _id: '2002-10-26', count: 1 },
{ _id: '2002-11-1', count: 1 },
{ _id: '2002-12-25', count: 1 },
{ _id: '2002-12-27', count: 1 },
{ _id: '2002-12-3', count: 1 },
{ _id: '2002-12-31', count: 1 },
```

```
  { _id: '2002-2-14', count: 1 },
  { _id: '2002-2-25', count: 1 },
  { _id: '2002-2-3', count: 1 },
  { _id: '2002-3-12', count: 1 },
  { _id: '2002-3-15', count: 1 },
  { _id: '2002-3-20', count: 2 },
  { _id: '2002-6-1', count: 1 },
  { _id: '2002-6-15', count: 1 },
  { _id: '2002-6-24', count: 1 },
  { _id: '2002-6-27', count: 1 },
  { _id: '2002-6-9', count: 1 },
  { _id: '2002-7-27', count: 1 },
  { _id: '2002-7-29', count: 1 },
  { _id: '2003-10-13', count: 1 },
  { _id: '2003-10-5', count: 1 },
  { _id: '2003-11-8', count: 1 },
  { _id: '2003-4-1', count: 1 },
  { _id: '2003-4-20', count: 1 },
  { _id: '2003-4-21', count: 2 },
  { _id: '2003-4-6', count: 1 },
  { _id: '2003-5-18', count: 1 },
  { _id: '2003-5-27', count: 1 },
  { _id: '2003-5-31', count: 1 },
  { _id: '2003-6-4', count: 1 },
  { _id: '2003-6-9', count: 1 },
  { _id: '2003-8-8', count: 1 },
  { _id: '2003-9-22', count: 1 },
  { _id: '2003-9-5', count: 1 },
  { _id: '2004-1-26', count: 1 },
  { _id: '2004-1-28', count: 1 },
  { _id: '2004-1-6', count: 1 },
  { _id: '2004-10-15', count: 1 },
  { _id: '2004-10-20', count: 1 },
  ... 346 more items
]
```

Além disso, esse código gera o documento “resultado-busca-phoenix.csv” que será útil para montar os gráficos temporais e para mais tarde, em Python, separar os dados nos conjuntos de treino e teste.

2 - Gráficos

1. Observar o gráfico em barras da série temporal para o ano x de forma a investigar como se comporta a distribuição das visualizações;
2. Observar o gráfico de linha da evolução do número de observações ao longo do tempo (anos).

O código abaixo é uma API que lê o documento “resultado-busca-phoenix.csv” e o retorna em forma de objeto para o endpoint /dados.

```
const fs = require('fs');
const csvParser = require('csv-parser');
const express = require('express');
const ejs = require('ejs');
```

```
const cors = require('cors');

const app = express();
app.set('view engine', 'ejs');
app.use(express.urlencoded({ extended: true }));
app.use(cors({
    origin: '*'
}));
app.listen(3000, () => {
  console.log('Servidor iniciado na porta 3000.')
});

let resultado = [];

app.get('/', (req, res) => {
  res.render('index', { title: 'Index' });
});


app.get('/dados', (req, res) => {
  fs.createReadStream('resultado-busca-phoenix.csv')
  .pipe(csvParser())
  .on('data', (data) => resultado.push(data))
  .on('end', async () => {
    res.json({ resultado });
  });
});
```

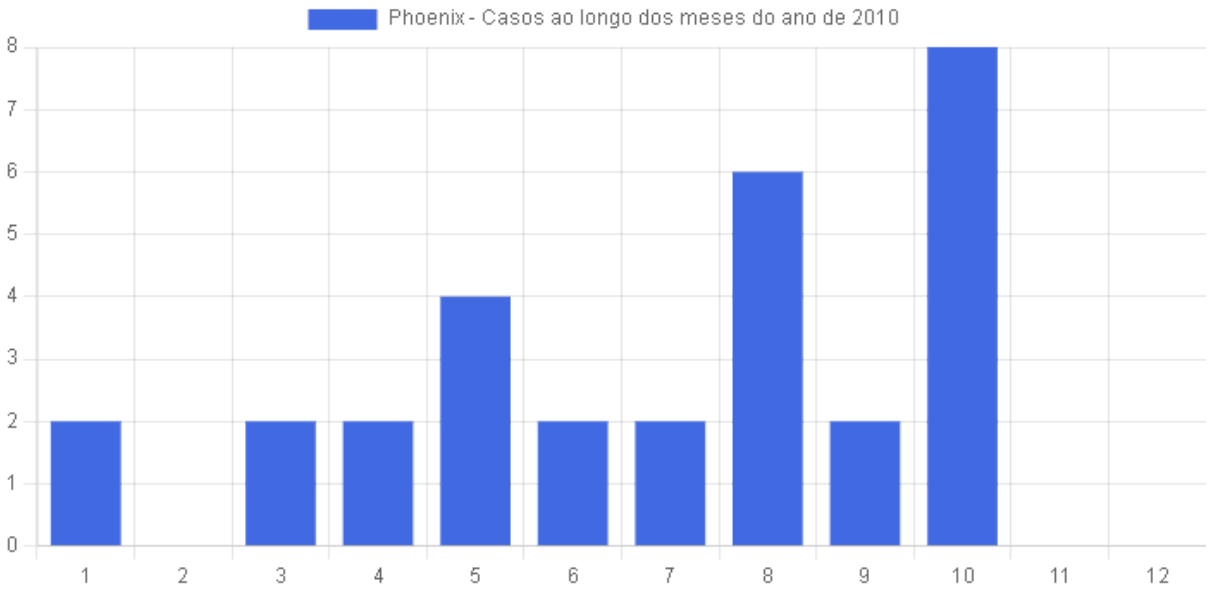
2.1 - Código

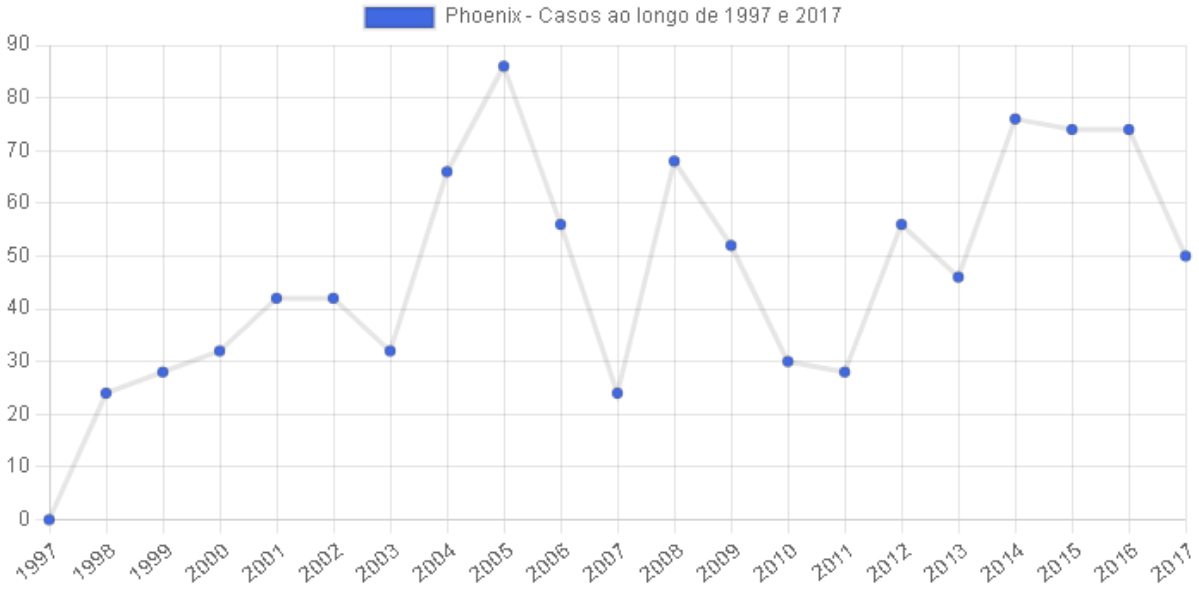
Assim, a requisição é feita ao endpoint /dados utilizando a função fetch nativa do JavaScript. Uma vez obtidos os dados, eles são filtrados manualmente, de modo a facilitar a construção dos gráficos, os quais são por fim montados com a biblioteca ChartJS. A seguinte função abaixo realiza esse trabalho dentro da tag HTML <script>.

```
async function req() {
  try {
    const res = await fetch('http://127.0.0.1:3000/dados', {
      method: 'GET'
    });
    const r = await res.json();

    console.log(r);

    // primeiro gráfico
    let anoAVerificar = 2010;
    let g1 = document.getElementById("grafico1").getContext("2d");

    let data = {
      labels: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
      datasets: [{
        label: `Phoenix - Casos ao longo dos meses do ano de
${anoAVerificar}`,
        backgroundColor: ["seagreen", "orange", "tomato", "teal",
"royalblue", "yellow", "lightgreen"],
```

```
        data: verificaTodosMesesAno(r.resultado, anoAVerificar)
      }]
    };

    let grafico1 = new Chart(g1, {
      type: 'bar',
      data: data,
    });

    // segundo gráfico
    let g2 = document.getElementById("grafico2").getContext("2d");

    saidaParaSegundoGrafico = verificaIntervaloAno(r.resultado, [1997,
2017]);

    let data2 = {
      labels: saidaParaSegundoGrafico.label,
      datasets: [{
        label: `Phoenix - Casos ao longo de 1997 e 2017`,
        backgroundColor: ["seagreen", "orange", "tomato", "teal",
"royalblue", "yellow", "lightgreen"],
        data: saidaParaSegundoGrafico.resultado
      }]
    };

    let grafico2 = new Chart(g2, {
      type: 'line',
      data: data2,
    });

  } catch(err) {
    console.log(err);
  }
}
```

2.2 - Resultado





3 - SARIMAX

Importações e dados csv para dataframe

```
import pandas as pd
import numpy as np
from pandas import DataFrame, Series
#!pip install pandasql
import pandasql
from sklearn.model_selection import train_test_split
# from statsmodels.tsa.statespace.sarimax import SARIMAX
import statsmodels.api as sm
import matplotlib.pyplot as plt
import statsmodels.api as sm
from pmdarima import auto_arima

df = pd.read_csv('resultado-busca-phoenix.csv')

df['_id'] = pd.to_datetime(df['_id'])
```

Separação em conjunto de teste e treino

```
# separa os dados em conjunto de treino e teste
treinamento, teste = train_test_split(df['count'], test_size=.30,
shuffle=False)
```

Obtendo o melhor modelo

```
# depois que vários modelos são criados, o melhor é selecionado
# execute esta parte do código separadamente, pois ela leva mais tempo
auto_arima_model = auto_arima(treinamento, d=1, D=1, seasonal=True,
trace=True, stepwise=False, max_p=8, max_q=8, max_d=8, max_P=8,
max_Q=8, max_D=8, start_p=0, start_q=0, start_d=0, start_P=0,
start_Q=0, start_D=0)
print(auto_arima_model)
```

Saída:

```
ARIMA(0,1,0)(0,0,0)[1] intercept   : AIC=568.957, Time=0.15 sec
 ARIMA(0,1,1)(0,0,0)[1] intercept   : AIC=inf, Time=0.94 sec
 ARIMA(0,1,2)(0,0,0)[1] intercept   : AIC=inf, Time=1.23 sec
 ARIMA(0,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=1.36 sec
 ARIMA(0,1,4)(0,0,0)[1] intercept   : AIC=inf, Time=2.22 sec
 ARIMA(0,1,5)(0,0,0)[1] intercept   : AIC=394.822, Time=1.49 sec
 ARIMA(1,1,0)(0,0,0)[1] intercept   : AIC=502.054, Time=0.10 sec
 ARIMA(1,1,1)(0,0,0)[1] intercept   : AIC=inf, Time=1.35 sec
```

```
 ARIMA(1,1,2)(0,0,0)[1] intercept   : AIC=inf, Time=1.21 sec
 ARIMA(1,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=1.55 sec
 ARIMA(1,1,4)(0,0,0)[1] intercept   : AIC=inf, Time=2.25 sec
 ARIMA(2,1,0)(0,0,0)[1] intercept   : AIC=469.117, Time=0.32 sec
 ARIMA(2,1,1)(0,0,0)[1] intercept   : AIC=inf, Time=2.14 sec
 ARIMA(2,1,2)(0,0,0)[1] intercept   : AIC=inf, Time=2.76 sec
 ARIMA(2,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=2.84 sec
 ARIMA(3,1,0)(0,0,0)[1] intercept   : AIC=454.238, Time=0.32 sec
 ARIMA(3,1,1)(0,0,0)[1] intercept   : AIC=inf, Time=1.92 sec
 ARIMA(3,1,2)(0,0,0)[1] intercept   : AIC=inf, Time=2.56 sec
 ARIMA(4,1,0)(0,0,0)[1] intercept   : AIC=446.411, Time=0.39 sec
 ARIMA(4,1,1)(0,0,0)[1] intercept   : AIC=inf, Time=2.02 sec
 ARIMA(5,1,0)(0,0,0)[1] intercept   : AIC=439.805, Time=0.54 sec

Best model:  ARIMA(0,1,5)(0,0,0)[1] intercept
Total fit time: 29.734 seconds
 ARIMA(0,1,5)(0,0,0)[1] intercept
```

Aplicando o melhor modelo ao SARIMAX

```
# o modelo superior é então aplicado ao sarimax
modelo_superior = sm.tsa.SARIMAX(treinamento,
order=auto_arima_model.order).fit()
```

Qualidade do modelo ajustado

```
# qualidade do modelo ajustado com AIC:
print("Qualidade AIC: ", modelo_superior.aic)
```

Saída:

```
Qualidade AIC:  391.81587521111993
```

Forecast

```
forecast = modelo_superior.forecast(steps=len(teste))
print('Forecast: \n', forecast)
```

Saída:

```
Forecast:
 312    1.089494
313    1.097377
314    1.096334
```

```
315    1.096842
316    1.096125
          ...
441    1.096125
442    1.096125
443    1.096125
444    1.096125
445    1.096125
Name: predicted_mean, Length: 134, dtype: float64
```

Erro médio e desvio padrão

```
from sklearn.metrics import mean_absolute_error as mae

# Calculando 'mean_absolute_error' com sklearn
print('Erro médio absoluto: ', mae(teste, forecast))

# Printando desvio padrão
print('Desvio padrão: ', np.std(forecast))
```

Saída:

```
Erro médio absoluto:  0.2028527344363802
Desvio padrão:  0.0005855728417645056
```

Considerações finais

Nesta Sprint foram criados os primeiros gráficos que permitiram uma melhor compreensão sobre os avistamentos de ovnis ao longo do tempo. Isso permitiu notar a existência de uma sazonalidade nos dados, de tal maneira que foi possível adotar um modelo de previsão temporal. Assim, dividiu-se os dados em dois conjuntos, treinamento e teste, para então aplicar o modelo SARIMAX.

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. Scikit Learn. **Scikit Learn**. Disponível em: <https://scikit-learn.org>. Acesso em: 02/08/2022
11. Stats Model. **Stats Model**. Disponível em: <https://statsmodel.org>. Acesso em: 02/08/2022
12. Py Pi. **pmdarima**. Disponível em: <https://pypi.org/project/pmdarima/>. Acesso em: 02/08/2022