INSTITUTO FEDERAL
Brasília
Campus Brasília

TECNOLOGIA EM SISTEMAS PARA INTERNET

Jorge Kayodê Lima Trindade
Marcos Rian Tomé de Oliveira
Nathália Teixeira Guimarães

RELATÓRIO DE PRÁTICA INTEGRADA DE CIÊNCIA DE DADOS E APRENDIZADO DE MÁQUINA

Brasília - DF

30/08/2021

Sumário

1. Objetivos

Tem-se como objetivo armazenar os dados no MongoDB, banco de dados não relacional, e ainda, fazer a análise de uma série temporal descritas nas atividades 3.2 e 3.4 do Canvas.

2. Descrição do problema

Após toda filtragem dos relatos que foram extraídos, estes ainda não possuem um ambiente para armazená-los. Devido a isto, é ideal um local para conduzir os dados e permitir que eles sejam acessados de forma flexível e escalável.

Ainda assim, depois de explorar os dados com gráficos e mapas, será levantada a análise da série temporal para avaliar o comportamento das variáveis ao longo do tempo, de forma que possa ser realizada uma projeção destas variáveis futuramente.

3. Desenvolvimento

Na etapa de armazenamento dos relatos obtidos, o grupo G2 fez o uso do banco de dados MongoDB, cuja função é armazenar e recuperar dados em formatos diferentes das tabelas. Este tipo de banco de dados permite fazer alterações sem ter que pré-definir a estrutura do banco de dados.

Para a análise, foi realizada a série temporal, a fim de que sejam observados todos os relatos e o que houve de crescimento ou decrescimento da variável analisada ao longo dos anos.

3.1 MongoDB

Inicialmente foi feita a importação dos pacotes e das bibliotecas utilizados durante todo processo de armazenamento.

Figura 1: Importação dos pacotes

```
from pymongo import MongoClient
from bson.son import SON
import pandas as pd
import pymongo
import pprint
import ssl
import dns
```

Fonte: Própria

Logo depois, foi criado uma função do atlas para realizar a conexão do aplicativo.

Figura 2: Função do atlas

```
#Função do atlas para conectar o aplicativo
client = pymongo.MongoClient(
    'mongodb+srv://Jorge:C9gbMv6WnAVq2Vh@ovni.fdmsu.mongodb.net/ovni?retryWrites=true&w=majority')
db = client.ovni
db
```

Fonte: Própria

Em seguida, foi feita a inserção do DataFrame ‘df_OVNI_preparado.csv’ como requerido na tarefa.

Figura 3: Inserção do DataFrame

```
#Inserção do DataFrame df_OVNI_preparado.csv como requisitado | não precisa mais ser rodado
dataOVNI = pd.read_csv('df_OVNI_preparado.csv', index_col=0)
dictOVNI = dataOVNI.to_dict('records')
dictOVNI
```

Fonte: Própria

Após isso, foi realizada a criação e a inserção do DataFrame na coleção ‘ovnis’ do banco de dados MongoDB.

Figura 4: Criação e inserção do DataFrame

```
#Criação e inserção do DataFrame na coleção ovnis do database MongoDB | ESSE CÓDIGO NÃO DEVE MAIS SER EXECUTADO
ovni = client["ovni"]
ovnis = ovni["ovnis"]
#Esse bloco de código SÓ DEVE SER RODADO UMA VEZ, pois fará o upload do DataFrame toda vez que for rodado
#x = coleção.insert_many(dictOVNI)
```

Fonte: Própria

E então, é utilizado um trecho de código para contar quantos documentos há na coleção.

Figura 5: Contagem dos documentos na coleção

```
#Contar quantos documentos há na coleção
doc_count = ovnis.count_documents({})
print("Há",doc_count,"documentos na coleção.")
```

```
Há 72875 documentos na coleção.
```

Fonte: Própria

A seguir, é realizado um resgate dos registros da coleção e ordenados pela função ‘SHAPE’.

Figura 6: Resgate dos registros

```
#Resgatar os registros da coleção e ordenar por SHAPE
doc_shapes = ovnis.find().sort("Forma",1)
for i in doc_shapes:
  print(i)
```

Fonte: Própria

Posteriormente, é feita a preparação de um dicionário a partir do DataFrame para fazer a query do banco de dados.

Figura 7: Preparação do dicionário

```
#Preparar um dicionário a partir do DataFrame para fazer o query do database
doc_states = db.ovnis.aggregate([
    { '$group': { '_id': "$Estado", 'count': { '$sum': 1 } } },
    { '$sort': { 'count': -1 } }
  ])
for i in doc_states:
  print(i)
```

Fonte: Própria

Depois, foi realizada uma verificação para saber quantas ocorrências existem por estado. Para isso, precisamos de uma soma de todos os relatos por estado.

Figura 8: Verificação das ocorrências por estado

```
#Verificar quantas ocorrências existem por estado
last = ovni.ovnis.find({},{"_id": 72875})
for i in last:
  print(i)
```

Fonte: Própria

Agora, foram verificadas quantas ocorrências foram registradas na cidade de Phoenix.

Figura 9: Verificação de quantas ocorrências na cidade de Phoenix

```
#Verificar quantas ocorrências existem da cidade de Phoenix
doc_phoenix = ovnis.find({'Cidade':'Phoenix'})
for i in doc_phoenix:
  print(i)
```

Fonte: Própria

Por último foi feita a busca das ocorrências no estado da Califórnia e uma ocultação dos seus ID’s.

Figura 10: Verificação das ocorrências no estado da Califórnia

```
#Buscar as ocorrências do estado da Califórnia e ocultar seus IDs
doc_cal_noID = ovnis.find({'Estado':'CA'},{'_id': 0})
for i in doc_cal_noID:
  print(i)
```

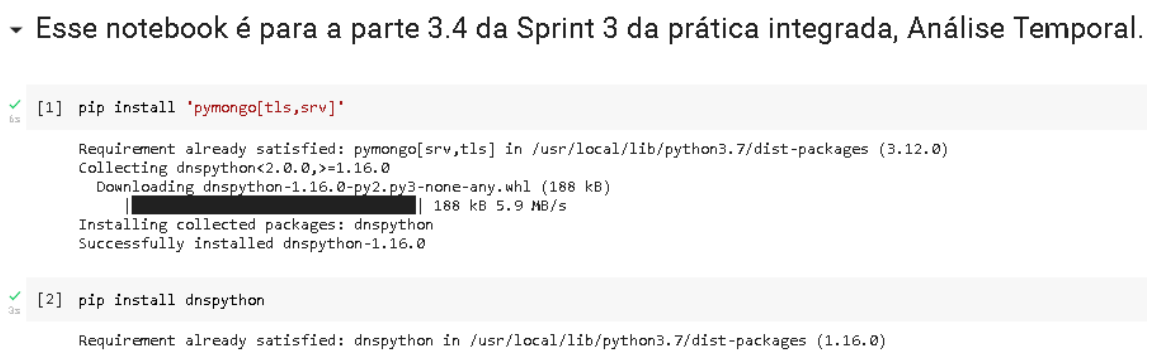
Fonte: Própria

3.2 Análise da série temporal

3.2.1 Recuperação dos Dados

Inicialmente foi feita a instalação do pymongo e da dnspython.

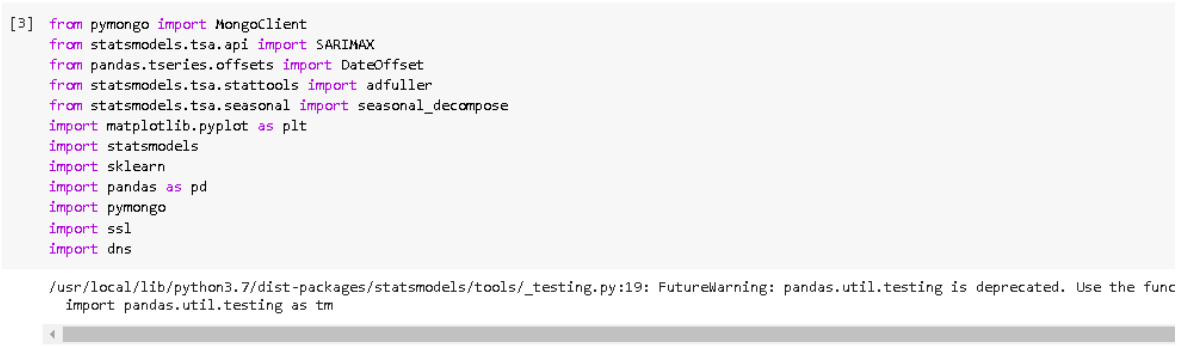
Figura 11: Instalação do ‘pymongo’ e do ‘dnspython’



Fonte: Própria

Em seguida, foram importados os pacotes e as bibliotecas a serem utilizadas durante todo processo da análise temporal.

Figura 12: Importação das bibliotecas



Fonte: Própria

Neste ponto, inserimos a função do Atlas para estabelecer a conexão com o MongoDB.

Figura 13: Função Atlas

Fonte: Própria

Aqui foi estabelecido o tamanho dos gráficos que seriam gerados ao longo do código.

Figura 14: Tamanho dos Gráficos Gerados

Fonte: Própria

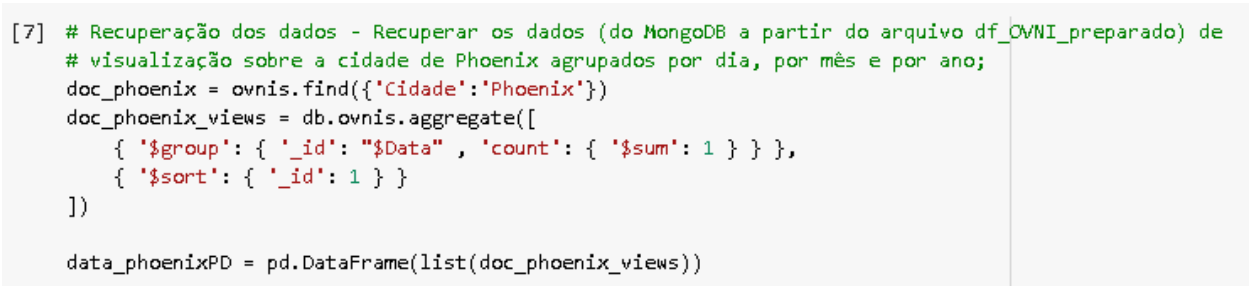
Logo após, foi criado o banco de dados denominado ‘ovni’ e a coleção chamada ‘ovnis’.

Figura 15: BD ‘ovni’ e coleção ‘ovnis’

Fonte: Própria

Em seguida, foi feita a recuperação dos dados do MongoDB a partir do arquivo ‘df_OVNI_preparado’. A recuperação dos dados foi sobre a cidade de Phoenix agrupadas por dia, por mês e por ano.

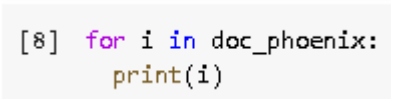
Figura 16: Recuperação de dados do MongoDB - Phoenix



Fonte: Própria

Aqui foi feito um laço ‘for’ para a leitura do filtro dos dados recuperados do dataframe ‘df_OVNI_preparado’.

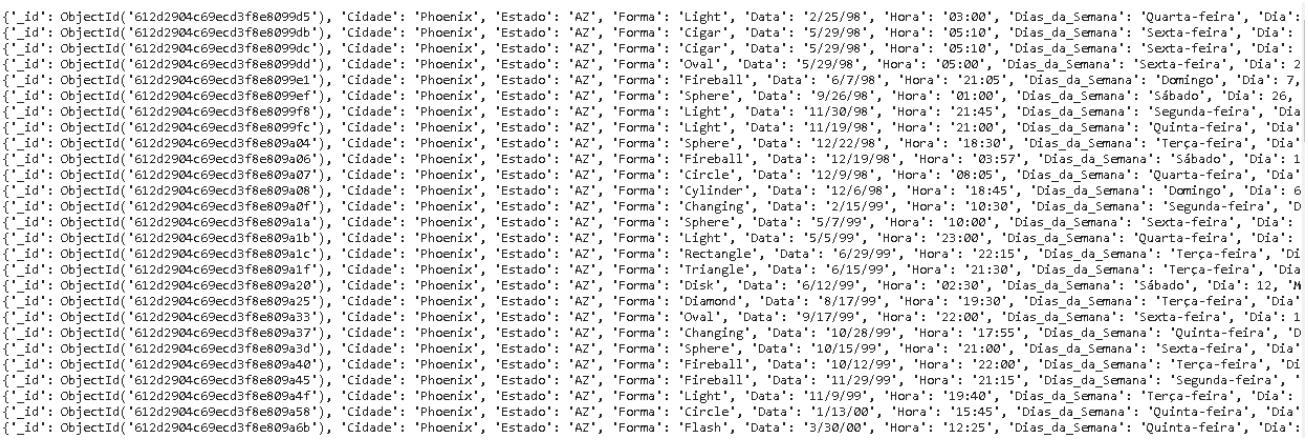
Figura 17: For para Impressão dos dados recuperados



Fonte: Própria

Na figura abaixo é apresentado o resultado do laço ‘for’ acima citado.

Figura 18: Debug do laço ‘for’



Fonte: Própria

Na figura a seguir, é mostrada a ordenação dos dados em ordem ascendente temporalmente.

Figura 19: Ordenação ascendente das informações

```
# Ordenar as observações de forma ascendente temporalmente (da observação mais
# antiga para a observação mais recente). Visualização dos dados em forma de Série Temporal:
data_phoenixPD['Data'] = pd.to_datetime(data_phoenixPD['_id'], format='%m/%d/%y')
data_phoenixPD.index = data_phoenixPD['Data']
data_phoenixPD
```

Fonte: Própria

Na figura abaixo, temos o resultado do código acima.

Figura 20: Resultado da ordenação ascendente das informações

Data	_id	count	Data
NaT	None	1	NaT
2000-01-01	1/1/00	14	2000-01-01
2001-01-01	1/1/01	16	2001-01-01
2002-01-01	1/1/02	6	2002-01-01
2003-01-01	1/1/03	17	2003-01-01
...	...	...	...
2015-09-09	9/9/15	16	2015-09-09
2016-09-09	9/9/16	12	2016-09-09
2017-09-09	9/9/17	13	2017-09-09
1998-09-09	9/9/98	1	1998-09-09
1999-09-09	9/9/99	6	1999-09-09

7219 rows × 3 columns

Fonte: Própria

3.2.2 Visualização dos dados em forma de série temporal

A seguir, foi feita a anulação das colunas com valores vazios, da coluna '_id' e da coluna 'Data' do dataframe 'data_phoenixPD'. Em seguida, o dataframe foi organizado conforme a coluna 'Data' que sobrou, já que esta estava duplicada.'

Figura 21: Tratamento e organização do Dataframe

```
[10] #Tratamento e organização final do DataFrame
     data_phoenixPD = data_phoenixPD.dropna()
     data_phoenixPD = data_phoenixPD.drop(columns=['_id','Data'])
     data_phoenixPD.sort_values(by='Data', kind='mergesort', inplace=True)
     data_phoenixPD
```

Fonte: Própria

Na figura abaixo, temos o resultado do código acima.

Figura 22: Resultado do Tratamento e organização do Dataframe

	count
Data	
1997-11-01	6
1997-11-02	1
1997-11-04	1
1997-11-07	1
1997-11-08	1
...	...
2017-10-27	11
2017-10-28	10
2017-10-29	7
2017-10-30	11
2017-10-31	6

7218 rows × 1 columns

Fonte: Própria

Função para saber o tamanho do dataframe.

Figura 23: Tamanho do dataframe ‘data_phoenixPD’

```
[11] len(data_phoenixPD)

     7218
```

Fonte: Própria

Na figura a seguir, foram estabelecidas as variáveis de tempo para a análise temporal.

Figura 24: Variáveis de tempo para a análise gráfica

```
[12] #Variáveis para escolher o período entre as datas
     start_date = '2002-12-31'
     end_date = '2003-03-30'
```

Fonte: Própria

A seguir, uma máscara para selecionar o período desejado para a análise temporal.

Figura 25: Máscara para filtrar o tempo de espaço da análise

```
[13] #Técnica utilizada para selecionar o período desejado
     mask = (data_phoenixPD.index > start_date) & (data_phoenixPD.index <= end_date)
     data2003 = data_phoenixPD.loc[mask]
     data2003
```

Fonte: Própria

Na figura abaixo, é trazido o resultado da máscara acima.

Figura 26: Resultado da máscara para filtrar o tempo de espaço da análise

	count
Data	
2003-01-01	17
2003-01-02	8
2003-01-03	1
2003-01-04	9
2003-01-05	4
...	...
2003-03-26	2
2003-03-27	2
2003-03-28	2
2003-03-29	5
2003-03-30	3

84 rows × 1 columns

Fonte: Própria

Em seguida, foram criados os eixos do gráfico da análise temporal.

Figura 27: Criação dos eixos

```
[14] #Criando os eixos
     X = data2003.index
     Y = data2003['count']
```

Fonte: Própria

Na figura a seguir, foi feita a plotação do gráfico em barras da série temporal para o ano de 2003.

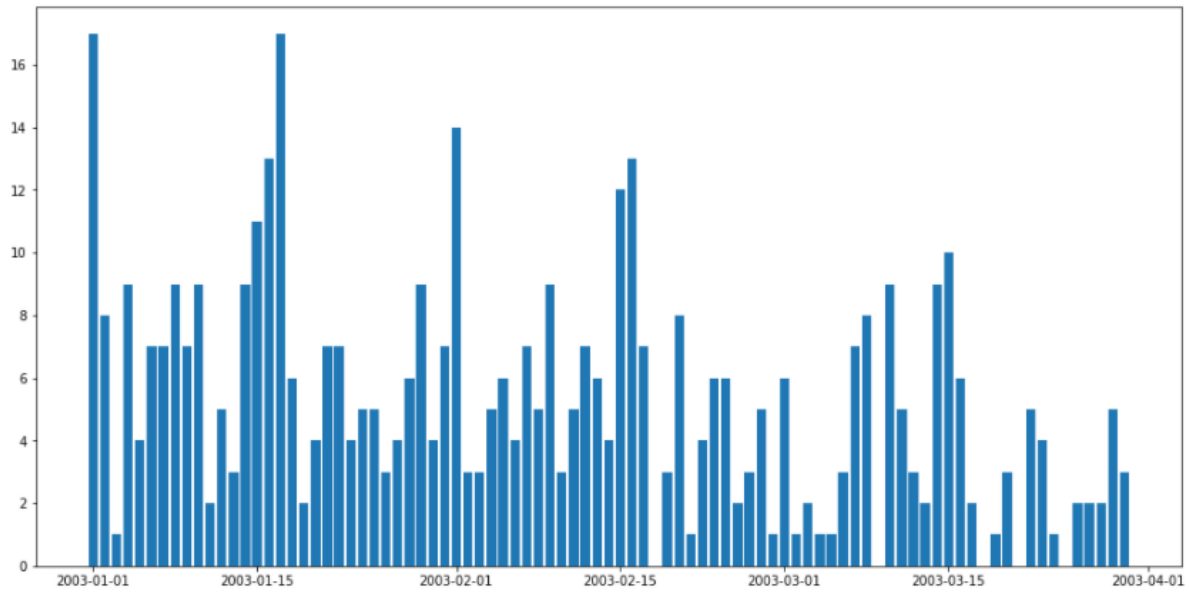
Figura 28: Geração do gráfico de barras

```
[15] # Observar o gráfico em barras da série temporal para o ano x de forma a investigar
     # como se comporta a distribuição das visualizações.
     plt.figure()
     barPlot = plt.bar(X,Y)
     plt.show()
```

Fonte: Própria

Na figura abaixo, é trazido o resultado do gráfico de barras.

Figura 29: Gráfico de Barras



Fonte: Própria

Na figura a seguir, foi feita a plotação do gráfico em linhas do número de observações ao longo dos anos.

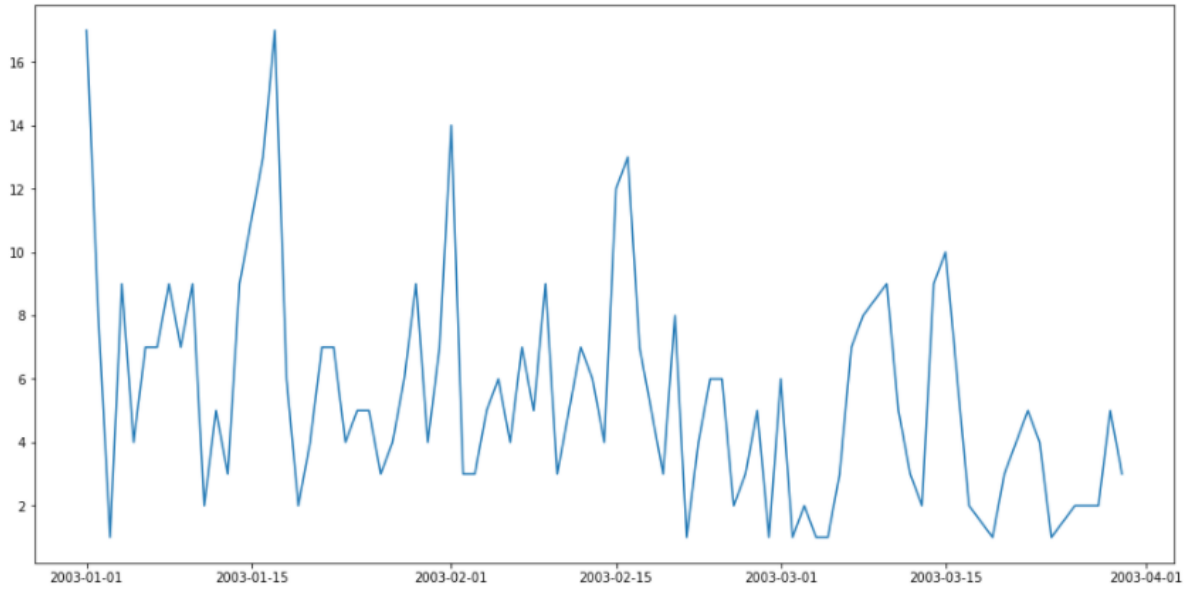
Figura 30: Geração do gráfico de linhas

```
[16] # Observar o gráfico de linha da evolução do número de observações ao longo do tempo (anos).
     plt.plot(X,Y)
     plt.show()
```

Fonte: Própria

A seguir, o resultado do gráfico de linhas.

Figura 31: Gráfico de Linhas



Fonte: Própria

Figura 32: Fechar janela de gráfico

```
[17] plt.close

     <function matplotlib.pyplot.close>
```

Fonte: Própria

3.2.3 Construção dos conjuntos de treinamento e teste

Na figura abaixo foi criado o dataset a partir do dataframe ‘data_phoenixPD’.’

Figura 33: Dataset

```
[18] # Construção dos conjuntos de Treinamento e Teste: Separar 70% das observações para
     # treinamento e 30% das observações para teste (como se trata de uma informação temporal,
     # não podemos pegar uma amostra aleatória, sugestão: calcular o índice que corresponde a 70%
     # das observações e considerar da primeira amostra até ele para treinamento; e do índice seguinte até o final para teste).
     #70% de 7218 = 5052
     #30% de 7218 = 2166
     dataset = data_phoenixPD.copy()
     dataset.sort_index(inplace=True)
     dataset
```

Fonte: Própria

A seguir, o resultado do dataset.

Figura 34: Resultado do Dataset

Data	count
1997-11-01	6
1997-11-02	1
1997-11-04	1
1997-11-07	1
1997-11-08	1
...	...
2017-10-27	11
2017-10-28	10
2017-10-29	7
2017-10-30	11
2017-10-31	6

7218 rows × 1 columns

Fonte: Própria

Em seguida, foi feita a divisão do conjunto de treinamento.

Figura 35: Conjunto de Treinamento.

```
[19] # Conjunto de treinamento:
     training = dataset.iloc[0:5052]
     training = training.asfreq("MS")
     training = training.fillna(0)
     training
```

Fonte: Própria

A seguir, o resultado do conjunto de treinamento.

Figura 36: Resultado do Conjunto de Treinamento

Data	count
1997-11-01	6
1997-12-01	6
1998-01-01	12
1998-02-01	2
1998-03-01	3
...	...
2011-07-01	30
2011-08-01	11
2011-09-01	21
2011-10-01	21
2011-11-01	22

169 rows × 1 columns

Fonte: Própria

E após, o conjunto de teste.

Figura 37: Conjunto de Teste

```
# Conjunto de teste:
test = dataset.iloc[5053:7218]
test = test.asfreq("MS")
test = test.fillna(0)
test
```

Fonte: Própria

Abaixo, é apresentado o conjunto de teste.

Figura 38: Resultado do Conjunto de Teste

Data	count
2011-12-01	18
2012-01-01	63
2012-02-01	24
2012-03-01	8
2012-04-01	13
...	...
2017-06-01	7
2017-07-01	14
2017-08-01	11
2017-09-01	11
2017-10-01	11

71 rows × 1 columns

Fonte: Própria

Abaixo são apresentados os dados decompostos sobre o dataset de treinamento.

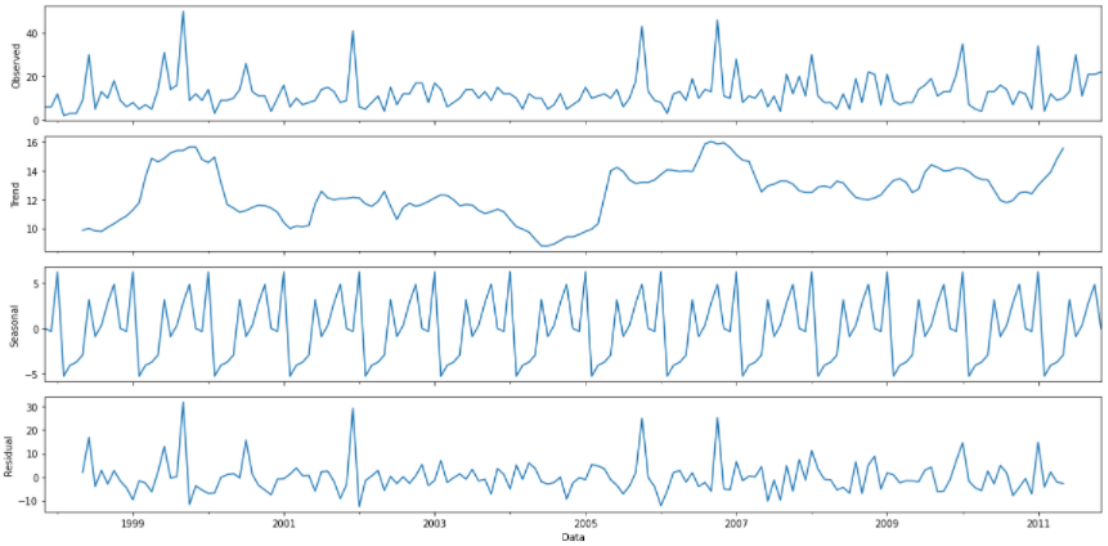
Figura 39: Dados decompostos do Dataset de Treinamento

```
#Gráficos com os dados decompostos sobre o datasets
decompose_data_train = seasonal_decompose(training, model="additive")
decompose_plot = decompose_data_train.plot()
```

Fonte: Própria

A seguir, o resultado dos dados decompostos sobre o dataset de treinamento.

Figura 40: Resultado dos Dados decompostos sobre o Dataset de treinamento



Fonte: Própria

Abaixo são apresentados os dados decompostos sobre o dataset de teste.

Figura 41: Dados decompostos do Dataset de Teste

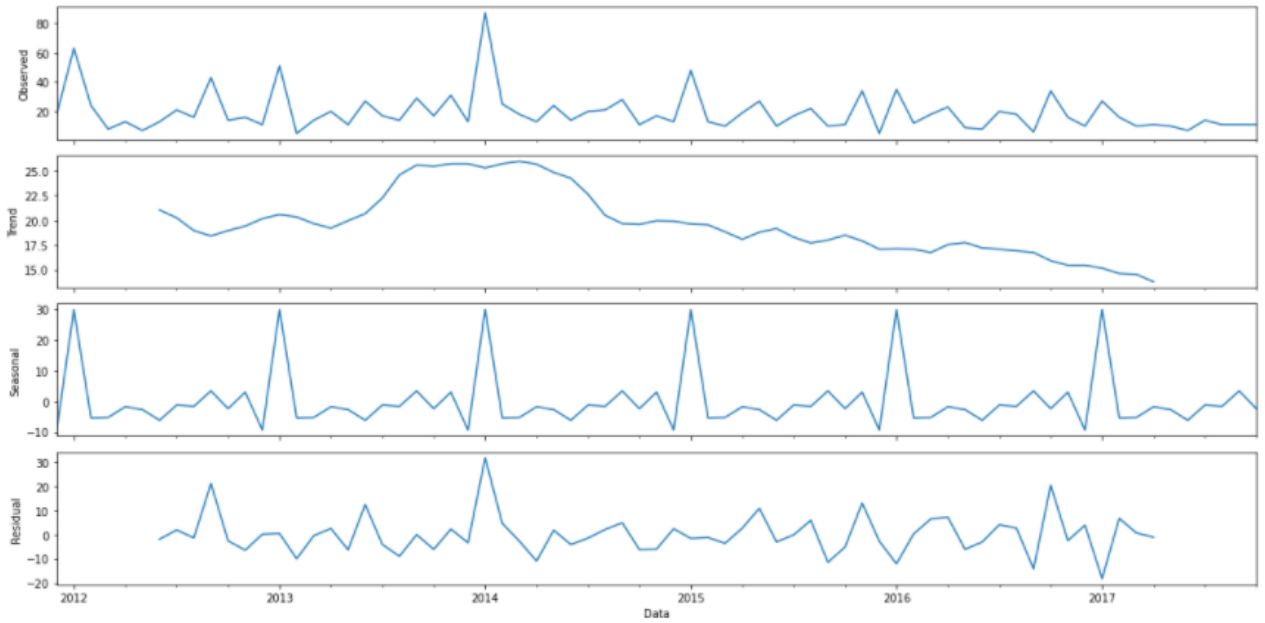
Figura 10:

```
#Gráficos com os dados decompostos sobre o datasets
decompose_data_test = seasonal_decompose(test, model="additive")
decompose_plot = decompose_data_test.plot()
```

Fonte: Própria

A seguir, o resultado dos dados decompostos sobre o dataset de teste.

Figura 42: Resultado dos Dados decompostos sobre o Dataset de teste



Fonte: Própria

3.2.4 Investigar os parâmetros para discriminar o melhor modelo

Neste ponto do código foi usado o ‘seasonal’ para gerar o gráfico com os dados decompostos do conjunto de treinamento na cor verde.

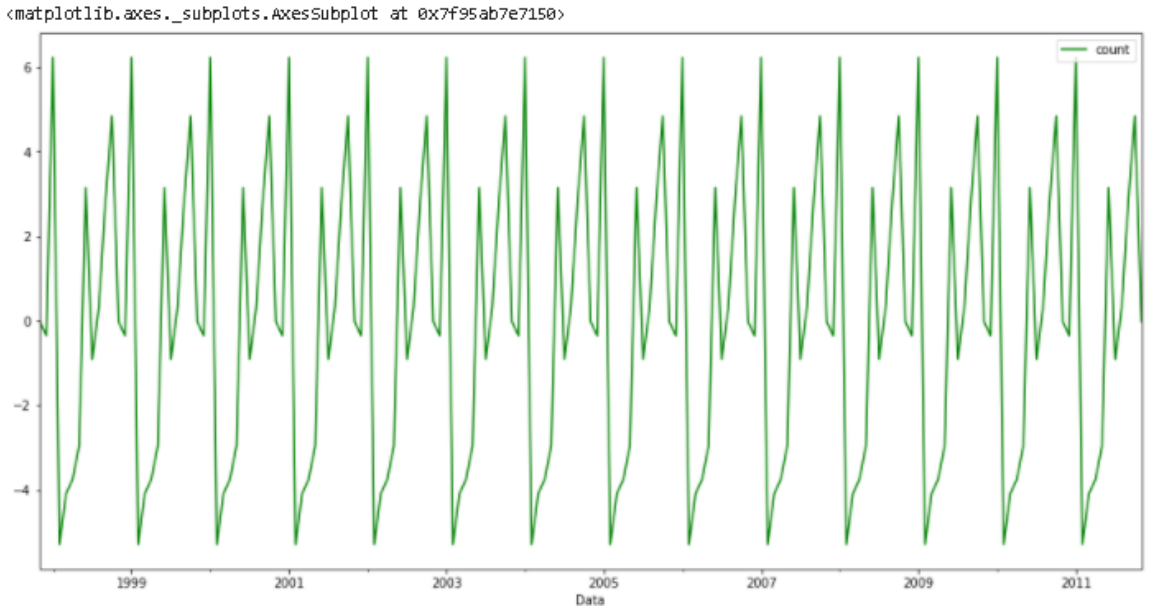
Figura 43: Gerar gráfico de sazonalidade

```
# Investigar os parâmetros para discriminar o melhor modelo: Utilizando o pacote
# statsmodels, vamos testar uma família de métodos apropriados para lidar com previsão
# de séries temporais chamados conjuntamente de SARIMAX (Links para um site externo.),
# ou seja, utilize a função SARIMAX para criar um modelo;
seasonality_train = decompose_data_train.seasonal
seasonality_train.plot(color='green')
```

Fonte: Própria

Na figura abaixo, é trazido o resultado do código acima.

Figura 44: Gráfico de sazonalidade do conjunto de treinamento



Fonte: Própria

Neste ponto do código foi usado o ‘seasonal’ para gerar o gráfico com os dados decompostos do conjunto de teste na cor vermelha.

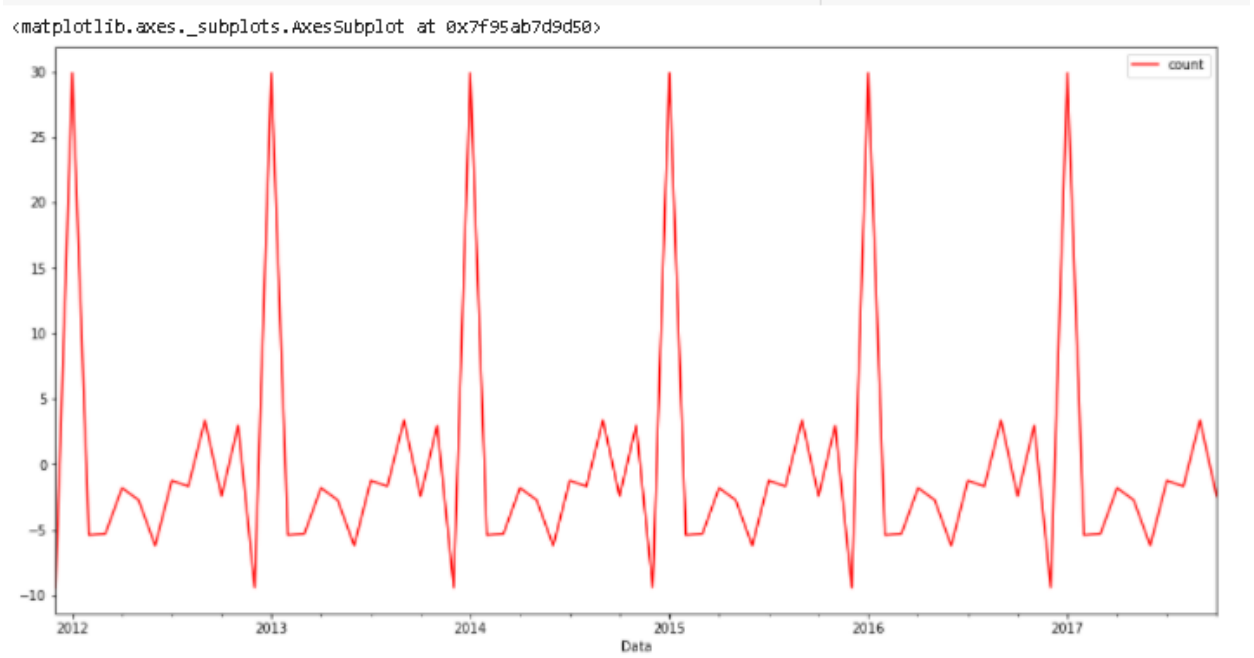
Figura 45:

```
[24] # Investigar os parâmetros para discriminar o melhor modelo: Utilizando o pacote
     # statsmodels, vamos testar uma família de métodos apropriados para lidar com previsão
     # de séries temporais chamados conjuntamente de SARIMAX (Links para um site externo.),
     # ou seja, utilize a função SARIMAX para criar um modelo;
     seasonality_test = decompose_data_test.seasonal
     seasonality_test.plot(color='red')
```

Fonte: Própria

Em seguida, é trazido o resultado do código acima.

Figura 46: Gráfico de sazonalidade do conjunto de teste



Fonte: Própria

A partir da análise dos gráficos acima, podemos concluir o disposto abaixo.

Figura 47: Diagnóstico obtido a partir dos gráficos de sazonalidade

A diferença no gráfico de sazonalidade é perceptível, deixando a entender que durante os 14 primeiros anos, relatos estavam sendo registrados durante o ano todo, mas nos últimos 5 anos, houve uma concentração de relatos no período da transição de ano.

Fonte: Própria

A seguir, a função ‘fit’ foi usada para ajustar o modelo de treinamento que foi testado anteriormente.

Figura 48: Função ‘fit’ para ajustar o modelo de treinamento

Modelos

```
[25] # Em seguida, chame a função fit() para ajustar o modelo; Best:(3,1,3,4)
     model = SARIMAX(training['count'], order=(0,0,1), seasonal_order=(3,1,3,4))
     results_train = model.fit()
     # Exemplo de qualidade do modelo de acordo com o parâmetro AIC:
     # A qualidade do modelo estimada pelo AIC é: 1085.2222824883293
     print(results_train.summary())
```

Fonte: Própria

Na figura a seguir, é trazido o ajuste do modelo de treinamento feito pelo código acima.

Figura 49: Resultado da função ‘fit’ para ajustar o modelo de treinamento

```
                                Statespace Model Results
==========================================================================================
Dep. Variable:                             count   No. Observations:                  169
Model:             SARIMAX(0, 0, 1)x(3, 1, 3, 4)   Log Likelihood                -569.910
Date:                           Wed, 01 Sep 2021   AIC                           1155.821
Time:                                   18:26:37   BIC                           1180.668
Sample:                             11-01-1997   HQIC                          1165.907
                                  - 11-01-2011
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.0195      0.098     -0.198      0.843      -0.212       0.174
ar.S.L4       -0.8590      0.167     -5.149      0.000      -1.186      -0.532
ar.S.L8       -0.9391      0.155     -6.069      0.000      -1.242      -0.636
ar.S.L12       0.0613      0.156      0.394      0.694      -0.244       0.366
ma.S.L4       -0.0660      0.206     -0.320      0.749      -0.470       0.338
ma.S.L8        0.0736      0.129      0.570      0.569      -0.180       0.327
ma.S.L12      -0.9242      0.138     -6.686      0.000      -1.195      -0.653
sigma2        51.4970      6.698      7.688      0.000      38.369      64.625
===================================================================================
Ljung-Box (Q):                       18.10   Jarque-Bera (JB):               263.90
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.68   Skew:                             1.83
Prob(H) (two-sided):                  0.16   Kurtosis:                         8.00
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

Fonte: Própria

A seguir, a função ‘fit’ foi usada para ajustar o modelo de teste que foi testado anteriormente.

Figura 50: Função ‘fit’ para ajustar o modelo de teste:

```
[26] # Em seguida, chame a função fit() para ajustar o modelo; Best:(3,1,3,4)
     model = SARIMAX(test['count'], order=(0,0,1), seasonal_order=(3,1,3,4))
     results_test = model.fit()
     # Exemplo de qualidade do modelo de acordo com o parâmetro AIC:
     # A qualidade do modelo estimada pelo AIC é: 1085.2222824883293
     print(results_test.summary())
```

Fonte: Própria

Na figura a seguir, é trazido o ajuste do modelo de teste feito pelo código acima.

Figura 51: Resultado da função ‘fit’ para ajustar o modelo de teste

```
                                Statespace Model Results
==========================================================================================
Dep. Variable:                             count   No. Observations:                   71
Model:             SARIMAX(0, 0, 1)x(3, 1, 3, 4)   Log Likelihood                -251.685
Date:                           Wed, 01 Sep 2021   AIC                            519.370
Time:                                   18:26:43   BIC                            537.008
Sample:                             12-01-2011   HQIC                           526.350
                                  - 10-01-2017
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          0.0026      0.156      0.017      0.987      -0.304       0.309
ar.S.L4       -1.4991      0.331     -4.536      0.000      -2.147      -0.851
ar.S.L8       -1.4564      0.268     -5.434      0.000      -1.982      -0.931
ar.S.L12      -0.5316      0.293     -1.812      0.070      -1.107       0.043
ma.S.L4        0.8640      1.582      0.546      0.585      -2.236       3.964
ma.S.L8        0.3304      1.723      0.192      0.848      -3.046       3.707
ma.S.L12      -0.4750      1.356     -0.350      0.726      -3.132       2.182
sigma2        77.7831    186.938      0.416      0.677    -288.608     444.174
===================================================================================
Ljung-Box (Q):                       30.92   Jarque-Bera (JB):                27.35
Prob(Q):                              0.85   Prob(JB):                         0.00
Heteroskedasticity (H):               0.72   Skew:                             0.59
Prob(H) (two-sided):                  0.44   Kurtosis:                         5.90
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

Fonte: Própria

Para finalizar, foi pedido que se realizasse o descrito abaixo.

Figura 52: Descrição da previsão com melhor modelo

A última etapa é realizar uma previsão utilizando o melhor modelo: Utilizando a função forecast sobre o modelo ajustado, faça uma previsão apropriada para a quantidade de dias que existem no seu conjunto de teste;

Fonte: Própria

A seguir, foi feita uma previsão sobre o modelo de treinamento utilizando a função 'forecast'.

Figura 53: Previsão sobre o melhor modelo

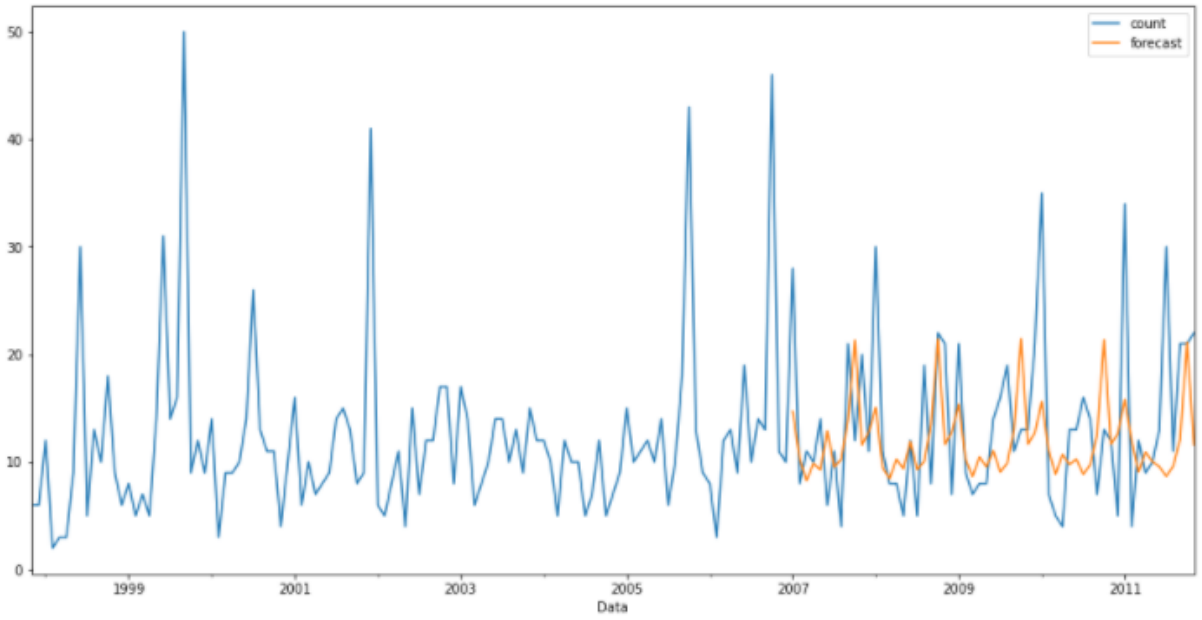
```
training['forecast'] = results_train.predict(start='2007-01-01',dynamic=True)
training[['count','forecast']].plot()
print("Previsão de nº de relatos para o próximo mês : ",results_train.forecast())
```

Fonte: Própria

Na figura abaixo, é trazido o resultado da previsão de números de relatos para o próximo mês (2011-12-01).

Figura 54: Resultado da previsão sobre o melhor modelo

```
Previsão de nº de relatos para o próximo mês :  2011-12-01    12.327269
Freq: MS, dtype: float64
```



Fonte: Própria

Figura 55: For para indexação do DataFrame

```
pred_date = [test.index[-1]+DateOffset(months=x)for x in range(0,24)]
pred_date = pd.DataFrame(index=pred_date[1:], columns=test.columns)
pred_date
```

Fonte: Própria

Figura 56: Resultado do Dataframe ‘pred_date’

	count
2017-11-01	NaN
2017-12-01	NaN
2018-01-01	NaN
2018-02-01	NaN
2018-03-01	NaN

Fonte: Própria

Concatenação do dataframe com a predição do conjunto de teste.

Figura 57: Concatenação do Dataframe ‘pred_date’ e ‘test_pred’

```
test_pred = pd.concat([test,pred_date])
#test_pred.drop(index)
test_pred
```

Fonte: Própria

Na figura abaixo segue o resultado do código acima.

Figura 58: Resultado da concatenação

	count
2011-12-01	18
2012-01-01	63
2012-02-01	24
2012-03-01	8
2012-04-01	13
...	...

Fonte: Própria

Abaixo é feita a plotação da previsão utilizando o ‘forecast’.

Figura 59: Previsão utilizando o ‘forecast’

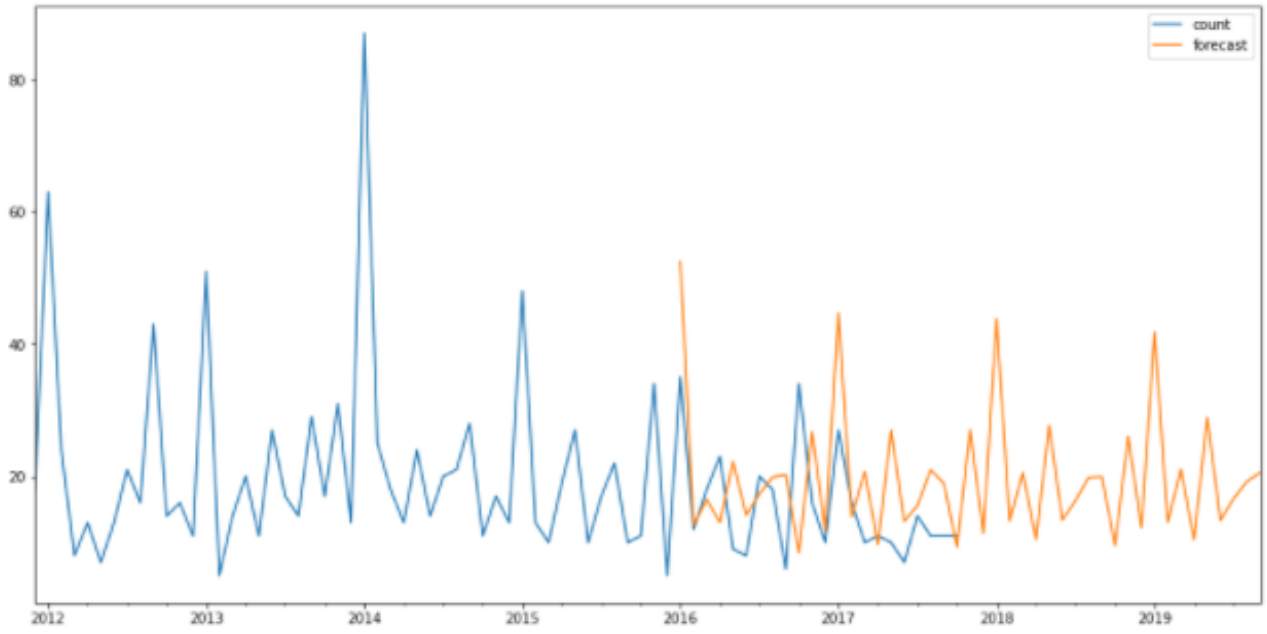
```
test_pred['forecast'] = results_test.predict(start='2016-01-01', end='2019-12-01',dynamic=True)
test_pred[['count','forecast']].plot()
print("Previsão de nº de relatos para o próximo mês : ",results_test.forecast())
```

Fonte: Própria

Na figura abaixo, é trazido o resultado da previsão de números de relatos para o próximo mês (2017-11-01).

Figura 60:

```
Previsão de nº de relatos para o próximo mês :  2017-11-01    22.258733
Freq: MS, dtype: float64
```



Fonte: Própria

Por fim, é trazido o cálculo do erro médio e o desvio padrão com relação ao conjunto de testes realizado.

Figura 61: Erro médio e Desvio-padrão

```
# Calcule o erro médio e o desvio-padrão com relação ao seu conjunto de testes.
desvio = test.std()[0]/100
erro = test.sem()[0]/10

print("O erro médio foi de: {:.1%} e o desvio-padrão foi de: {:.2%}".format(erro,desvio))
```

```
O erro médio foi de: 16.2% e o desvio-padrão foi de: 13.61%
```

Fonte: Própria

4. Considerações finais

Houve mais dificuldade no desenvolvimento desta sprint do que das demais devido ao uso de bibliotecas ainda não exploradas pelos integrantes do grupo, assim como da plataforma MongoDB. Felizmente, a equipe conseguiu finalizar mais esta etapa ainda assim.

Apenas os alunos Jorge, Marcos e Nathália participaram do desenvolvimento da sprint 3.

Referências

<https://www.statsmodels.org/stable/index.html> Acessado em: 31 de agosto de 2021.
<https://analyticsindiamag.com/complete-guide-to-sarimax-in-python-for-time-series-modeling/ > Acessado em: 31 de agosto de 2021.