INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Brasília
Campus Brasília

TECNOLOGIA EM SISTEMAS PARA INTERNET

Matheus Nícollas de Souza Mota
Thiago Marinho da Silva Campos

RELATÓRIO DE PRÁTICA INTEGRADA
DE
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

Brasília - DF
31/10/2020

Sumário

1. Objetivos

O objetivo é utilizar um modelo para estimar a quantidade de visualizações de óvnis que teremos no futuro.

2. Descrição do problema

O projeto consiste em reunir fatos interessantes relacionados a OVINIs, a partir de relatos realizados dentro de um período de vinte anos usando o site Nuforc. O desafio consiste em fazer a extração de dados de forma tabular, afinal, para que os dados possam ser analisados eles acabam se tornando tabelas.

O WebScraping consiste em extrair os dados formatados com tag's da linguagem HTML. Como iremos extrair vinte anos de dados, consultaremos 240 páginas web, uma por cada mês, por vinte anos, entre setembro 1997 e agosto de 2017.

Para fazer análises é necessário que os dados estejam apresentados de forma ordenada e de uma maneira funcional para que facilite a realização de pesquisas.

Após o tratamento de dados precisamos armazena-los, e isto será feito utilizando o MongoDB.

Nesta ultima etapa precisaremos recuperar os dados de visualização sobre a cidade de Phoenix, orderanaremos de forma ascendente temporalmente, construiremos um conjunto de testes, investigaremos parâmetros para discriminar o melhor modelo, mediremos a qualidade do modelo ajustado, para por fim poder realizar uma previsão utilizando o melhor modelo.

3. Desenvolvimento

O desenvolvimento do algoritmo foi feito na plataforma Google Collab, esta plataforma foi escolhida pois ao iniciar um notebook na mesma, as bibliotecas e dependências do Python são todas da nuvem.

Resgatamos os dados através do MongoDB. Foi feita uma ordenação sobre as observações de forma ascendente temporalmente. Foi criado um novo dataframe com os dados da cidade de Phoenix, agrupamos as datas e excluímos colunas não necessárias.

Investigamos como se compota a distribuição das visualizações a partir da série temporal implementada em um gráfico de barras. Também foi implementado e observado um gráfico de linhas exibindo dados sobre a evolução do número de observações ao longo do tempo(anos).

Na terceira etapa foi separado os os dados com 70% das observações para treinamento e 30% para teste. Utilizamos o pacote statsmodel para testar o modelo de métodos apropriados para lidar com a previsão de séries temporais.

Na ultima etapa foi realizado uma previsão utilizando o modelo mais apropriado para este caso.

4 Código implementado

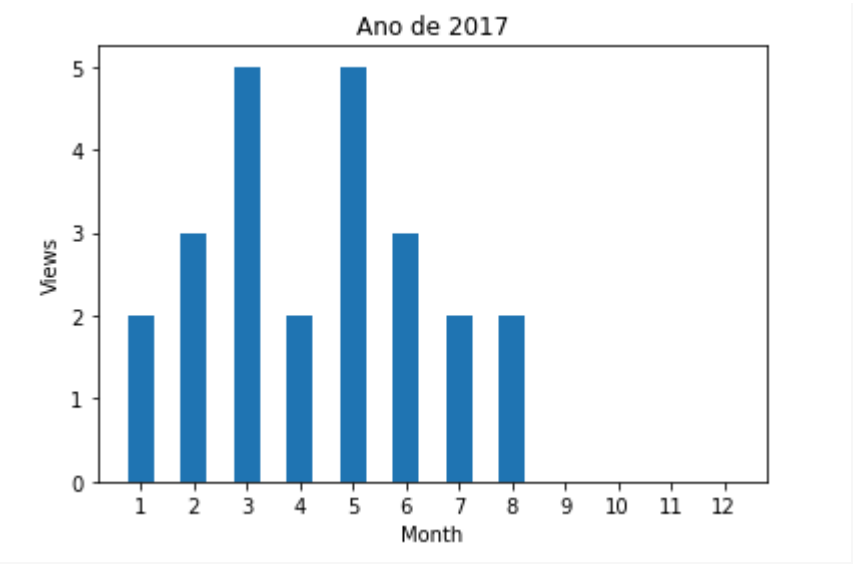
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#Ordenar as observações de forma ascendente temporalmente (da observaç
ão 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'].trans
form('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')
```

	Sight_Date	Views
335	1999-06-12	1
970	2001-11-12	1
1249	2003-05-31	1
1781	2005-02-21	1
1793	2005-03-20	1
...	...	...
70762	2017-06-15	2
71348	2017-07-06	1
71057	2017-07-26	1
71857	2017-08-04	1
71716	2017-08-14	1

305 rows × 2 columns

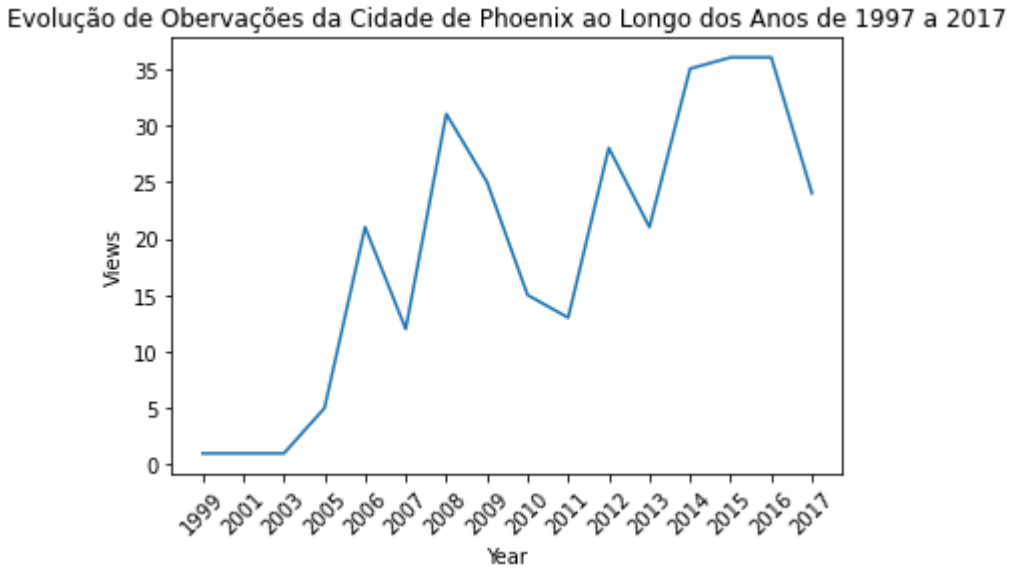
```
#Atribui o daframe a uma outro para manipulção mais livre
df_mes = df_ovnis
#Define o ano que quer ser visto no grafico
ano = 2017

#Transforma a coluna 'Sight_Date' em formato de data para separar o an
o
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'].transfo
rm('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 observacoes 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 da
frame 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 an
o
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'].transfo
rm('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 Obervações da Cidade de Phoenix ao Longo dos An
os de 1997 a 2017")
plt.xticks(rotation=45)

plt.show()
```




```
# Transforma 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 para ficar saltado no Datafram
e.
df_views.set_index('Sight_Date', inplace = True)
df_views
```

Sight_Date	Views
1999-06-12	1
2001-11-12	1
2003-05-31	1
2005-02-21	1
2005-03-20	1
...	...
2017-06-07	1
2017-07-26	1
2017-07-06	1
2017-08-14	1
2017-08-04	1

305 rows × 1 columns

```
df_views.head()
```

Sight_Date	
1999-06-12	1
2001-11-12	1
2003-05-31	1
2005-02-21	1
2005-03-20	1

```
# Isso separa os conjunto df_views no conjunto de treinamento (df_trai
n) e no conjunto de teste (df_test).
# O conjunto df_train possui 70% dos dados do conjunto df_views, enqua
nto o df_test possui 30% dos dados.
df_train, df_test = df_views.iloc[:213, :], df_views.iloc[213:, :]
```

```
df_train.shape
```

```
(213,1)
```

```
df_train.head()
```

Sight_Date	
1999-06-12	1
2001-11-12	1
2003-05-31	1
2005-02-21	1
2005-03-20	1

```
df_test.shape
```
```
(92,1)
```

```
df_test.head()
```

Sight_Date	
2015-01-06	2
2015-02-18	1
2015-02-11	2
2015-02-11	2
2015-03-20	1

```
import numpy as np
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df_views.drop('Vie
ws',axis=1), df_views['Views'], test_size=0.3, random_state=0, shuffle
=False)
x_train.shape, y_train.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()
```

Statespace Model Results

Dep. Variable:	Views	**No. Observations:**	213
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-191.189
Date:	Wed, 28 Oct 2020	**AIC**	386.378
Time:	23:36:52	**BIC**	393.101
Sample:	0 - 213	**HQIC**	389.095
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9235	0.013	69.757	0.000	0.898	0.949
sigma2	0.3493	0.008	42.421	0.000	0.333	0.365

Ljung-Box (Q):	48.28	**Jarque-Bera (JB):**	15669.28
Prob(Q):	0.17	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.20	**Skew:**	0.59
Prob(H) (two-sided):	0.00	**Kurtosis:**	45.00

```
#Cria Array vazios que vao receber os valores a serem calculados
array_forecast = []
array_test = []
diferenca = []

#for que intera as visualizações do dataframe forecast no array 'array
_forecast'
for i in df_forecast.itertuples():
  array_forecast.append(float(df_forecast.Views[i.Index]))

#for que intera as visualizações do dataframe test no array 'array_tes
t'
for index, row in df_test.iterrows():
  teste = str(row["Views"])
  array_test.append(float(teste))

#calcula a diferença de cada elemnto do array
for i in range(92):
  diferenca.append(array_forecast[i] - array_test[i])

#printa todas as diferenças de cada linha do dataframe, mas no formato
 de array
print(diferenca)
```

```
[-0.15307858398886953, 0.7055593584602797, -0.4249829472907525, -
0.5455336373842796, 0.3431425369913923, 0.2403393581624622,
0.14540466185587575, 0.057736199990302106, -0.023222179873839455, -
0.09798406266216309, -0.16702372387369757, -0.2307791382965656, -
0.2896547584386955, -0.34402408029941345, -0.39423201275869457, -
0.44059706561503786, -0.4834133701524721, -0.522952545054793, -
0.5594654195040463, -0.5931836243942719, -0.6243210617548763, -
```

```
0.6530752617053791, -0.679628635549804, -1.704149632960113, -
1.7267938105896397, -4.747704818895606, -4.767015313430944, -
4.784847796386485, -4.801315393722106, -4.816522572816809, -
0.8305658051903708, -0.8435341785007469, -0.8555099616996273, -
0.8665691269313867, -0.8767818314862575, -0.8862128628651504, -
0.8949220497795254, -0.902964641693627, -0.9103916593168231, -
0.9172502182695096, -0.9235838279758553, -0.9294326676795083, -
1.9348338413332546, -1.9398216129796044, -1.9444276241155125, -
1.9486810944201598, -0.9526090071191701, -0.956236280161182, -
0.959585924292688, -0.9626791890339351, -1.9655356974819358, -
1.9681735707957506, -0.9706095431537548, -0.9728590679121594, -
0.9749364156382317, -0.9768547646401242, -0.9786262845676137, -
0.980262213614099, -0.9817729298096133, -0.9831680168571184, -
0.9844563249297368, -0.9856460268146063, -0.9867446697595235, -
1.987759223351282, -1.9886961237294367, -1.9895613144159785, -
1.990360284019932, -0.9910981010560738, -0.9917794460986479, -
0.9924086414740595, -0.9929896786809109, -0.9935262437113275, -
0.994021740434207, -0.9944793121887318, -1.9949018617251286, -
1.995292069619177, -0.9956524112772842, -0.9959851726400039, -
0.9962924646836179, -0.9965762368117781, -0.9968382892221612, -
0.9970802843265881, -0.9973037572970561, -0.9975101258045846, -
0.9977006990126569, -1.9978766858823103, -1.9980392028415601, -
0.9981892808678118, -0.9983278720281902, -0.9984558555192766, -
0.9985740432445683, -0.9986831849650437]
```

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

```
Erro Médio:
1.1942378489196437
```

```
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)
```

Github: https://github.com/Prof-Fabio-Henrique/pratica-integrada-icd-e-iia-2020-1-g13-mmt

5 Considerações Finais

Com esta etapa do projeto podemos fechar o ciclo no tratamento dos dados, depois de preparados, armazenados em bancos de dados, após as análises, podemos criar formas de investigar os parâmetros dos dados e encontrar o melhor modelo de predição para ser aplicado sendo assim possível realizar uma previsão de sobre um novo aparecimento de óvni.

6 Referencias

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: 28/10/2020

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: 28/10/2020

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: 28/10/2020

Stick-Learn; Machine Learning in Python. Disponível em: https://scikit-learn.org/stable/ . Acesso em: 28/10/2020