PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

José Clovis von Zuben Filho

PREVISÃO DE RESERVAS HIDROGRÁFICAS PARA PRODUÇÃO DE ENERGIA ELÉTRICA NO BRASIL
MODELOS ARIMA E LSTM

Rio de Janeiro
2021

José Clovis von Zuben Filho

PREVISÃO DE RESERVAS HIDROGRÁFICAS PARA PRODUÇÃO DE ENERGIA ELÉTRICA NO BRASIL MODELOS ARIMA E LSTM

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Rio de Janeiro
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

Um dos maiores desafios da sociedade para obter uma melhor qualidade de vida, se dá pela previsibilidade de eventos futuros. Na história temos desde o pequeno agricultor em sua lavoura se arriscando com as intemperes do tempo e ataques de possíveis pragas, até os dias atuais em que a falta de chuva impede a geração de energia elétrica e provisiona-se termoelétricas como sistema de retaguarda.

Outrora os problemas em administrar negócios, eram resolvidos pela expertise de longa data de um profissional, que passou por diversas situações até amadurecer na previsão de seus atos. Assim os eventos futuros eram baseados em “feelings” e empirismo técnico e não por dados concretos.

Na década de 1970 houve a introdução de Sistemas Especialistas que simulavam o raciocínio de um profissional, codificando regras de inferência e incertezas, mas com o decorrer do tempo demonstrou-se ser altamente restrito na área de domínio do problema a ser resolvido. A evolução na aquisição de dados com os sistemas de base de dados de Big Data, viabilizaram uma categoria dos Sistemas Especialistas de Previsão, a partir dos dados históricos observados, assim nasce a Ciência de Dados.

Agora o profissional especialista não requer mais confiar no seu “feeling” para prever situações futuras, a Ciência de Dados com suas ferramentas e modelos assume este papel, deixando ao profissional o papel de controle de qualidade.

Este trabalho foca na previsão das reservas de água nos reservatórios brasileiros, analisando séries temporais, para que haja tempo de provisionar recursos antes do desabastecimento.

1.2. O problema proposto

A energia elétrica é um dos bens mais utilizados em nossa vida cotidiana, sem ela hospitais, prédios e casas pararíam, a vida foi moldada pela energia elétrica, e seu consumo cresce vertiginosamente, mesmo com diversas fontes de energia, a de usina hidroelétrica responde por aproximadamente 90% no Brasil, seu uso precisa ser muito bem controlado.

Os dados deste trabalho são da empresa ONS (Operador Nacional de Sistema Elétrico), que tem a função principal de controlar os níveis dos reservatórios de água em todo o Brasil, utilizando sensoriamento remoto e internet para disponibilizar os dados histórico.

O principal objetivo deste trabalho é de analisar as séries temporais (data, profundidade em metros) no período de 01/01/2003 até 01/02/2021 de três reservatórios da base de dados, são eles: Furnas, Sobradinho, Belo Monte.

Como o sensoriamento é remoto, sem o fator humano para introduzir erros, faz com que o aspecto de coleta geográfico seja facilmente contornado.

2. Coleta de Dados

Os dados da ONS são obtidos direto no site da empresa, pelo link http://www.ons.org.br, mas também estão disponíveis não tão completos no link http://dados.gov.br . A extração dos dados é feita de forma manual, entrando no site da empresa, navegando até a seção "*Resultado da Operação→Histórico da Operação→Dados Hidrológicos/Níveis*", nela temos como na figura 1 a opção "Download" que é a extração após preenchimento das informações: escala em dia, período, nome do reservatório. Nesta tela encontra-se a opção de extração "Crosstab" do gráfico temporal, que tem o formato de arquivo .CSV mais condizente com a leitura que será feita pela biblioteca Pandas do Python.

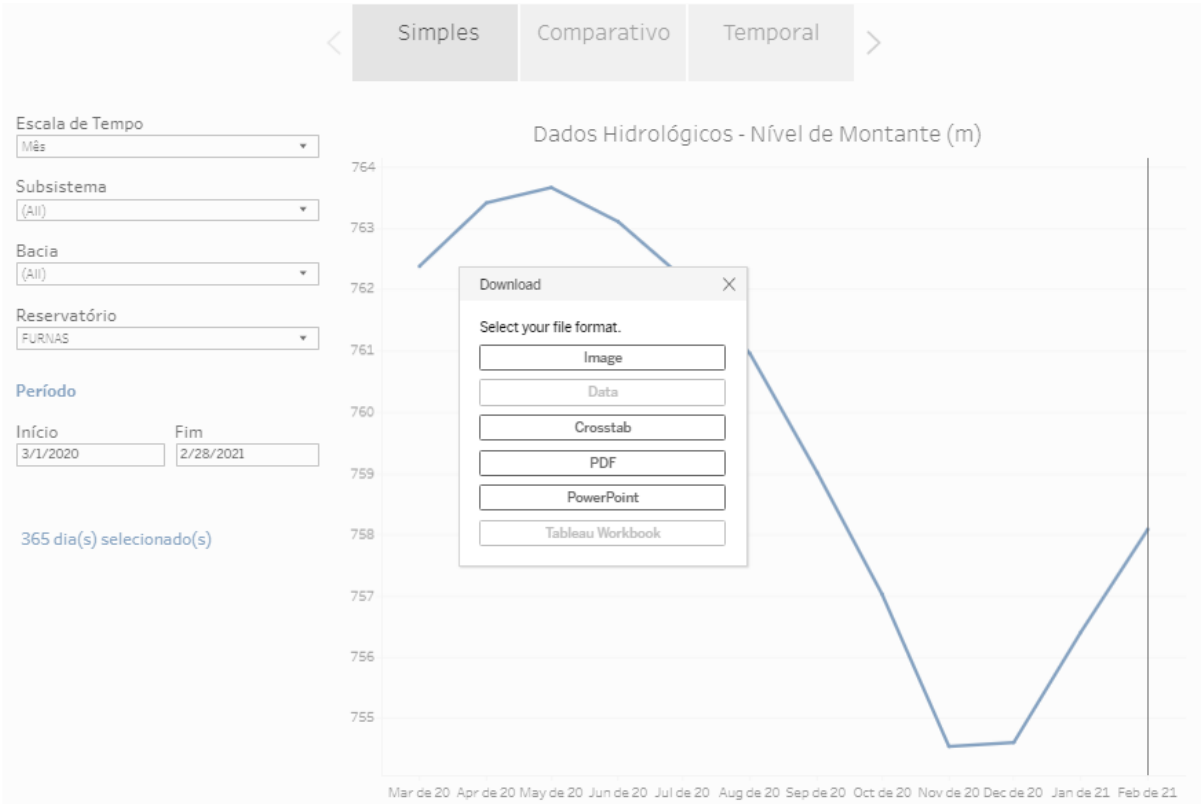

Figura - 1

O arquivo <nome do reservatório>.CSV, vem no formato Unicode padrão UTF-16, com os campos separados por TAB. Duas linhas de cabeçalho devem ser ignoradas na leitura, o valor da profundidade do reservatório vem com vírgula no padrão brasileiro bem como as datas. Para nosso trabalho somente a primeira coluna “data” e a última “profundidade” são necessárias, as demais informações são ignoradas. O arquivo gerado contém duas séries identificadas pela posição onde está a data, se a data estiver na primeira coluna são os valores da primeira série, se estiver na segunda coluna são os valores da segunda série.

Nome da coluna/campo	Descrição	Tipo
data1	Data Série #1	Datetime((DD/MM/AAAA)
data2	Data Série #2	Datetime((DD/MM/AAAA)
data_inicio_semana	Data do Início da Semana	Datetime((DD/MM/AAAA)
---	Campo não usado	---
---	Campo não usado	---
tipo_extracao	Dia, Mês, Semana	String
Nome	Nome do reservatório	String
profundidade	Profundidade em metros	Float

3. Processamento/Tratamento de Dados

O processo de extração dos três reservatórios é idêntico, entretanto determinados reservatórios iniciaram suas atividades em anos diferentes, assim vale verificar o gráfico da Figura – 1, para descobrir a data inicial correta.

Uma extração no período de 18 anos, nos dá um volume de amostras entorno de 6.600 registros, mas o tamanho do arquivo não passa de umas centenas de KBytes.

Ao final do processo de extração manual teremos os seguintes datasets:

Nome da coluna/campo	Descrição
Furnas.csv	Série com os níveis de profundidade do reservatório de Furnas para treinamento.
Sobradinho.csv	Série com os níveis de profundidade do reservatório de Sobradinho para treinamento.
BeloMonte.csv	Série com os níveis de profundidade do reservatório de Belo Monte para treinamento.

Os datasets não tem qualquer campo incompleto ou informação que deva ser tratada/sanitarizada antes do treinamento, assim o comando da Figura - 2 é o suficiente para importação dos dados no Python.

```
import pandas as pd
dataFrame = pd.read_csv("furnas.csv", index_col=0, header=2, delimiter='\t',
                                encoding='utf-16', usecols=[0,6,8],
                                 names=['data','nome','profundidade'],
                                parse_dates=['data'], dayfirst=True, decimal=',')
print (dataFrame)

Resultado:
data       nome     profundidade
2003-03-01  FURNAS   767.340000
2003-03-02  FURNAS   767.350000
2003-03-03  FURNAS   767.370000
         ...         ...
2021-02-26  FURNAS   757.909973
2021-02-27  FURNAS   758.000000
2021-02-28  FURNAS   758.080017
[6575 rows x 2 columns]
```

Figura – 2

4. Análise e Exploração dos Dados

Para analisar as tendências das séries temporais, valeu-se do código em Python da Figura – 3 abaixo, para imprimir os gráficos dos três reservatórios.

```
df.plot (color='r', title='Reservatório xxx', y=['profundidade'],
       xlabel='Data', linewidth=1, kind='line')
plt.grid()
plt.show()
```

Figura – 3

São eles:

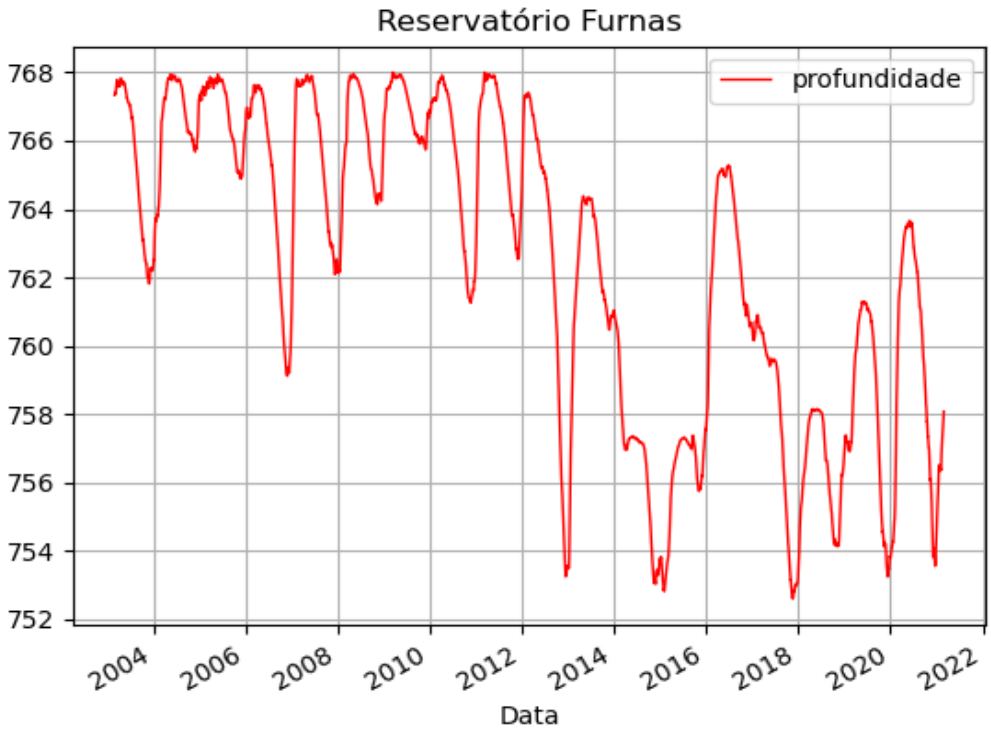

Figura – 4

Este gráfico demonstra o fator sazonalidade da série temporal, bem como uma tendência de queda permanente da profundidade aferida a partir do ano 2014, que leva a crer que houve algum fato histórico que justifique a perda de aproximadamente quatro metros no total do reservatório; desta forma o comportamento dos métodos preditivos deve modelar esta queda.

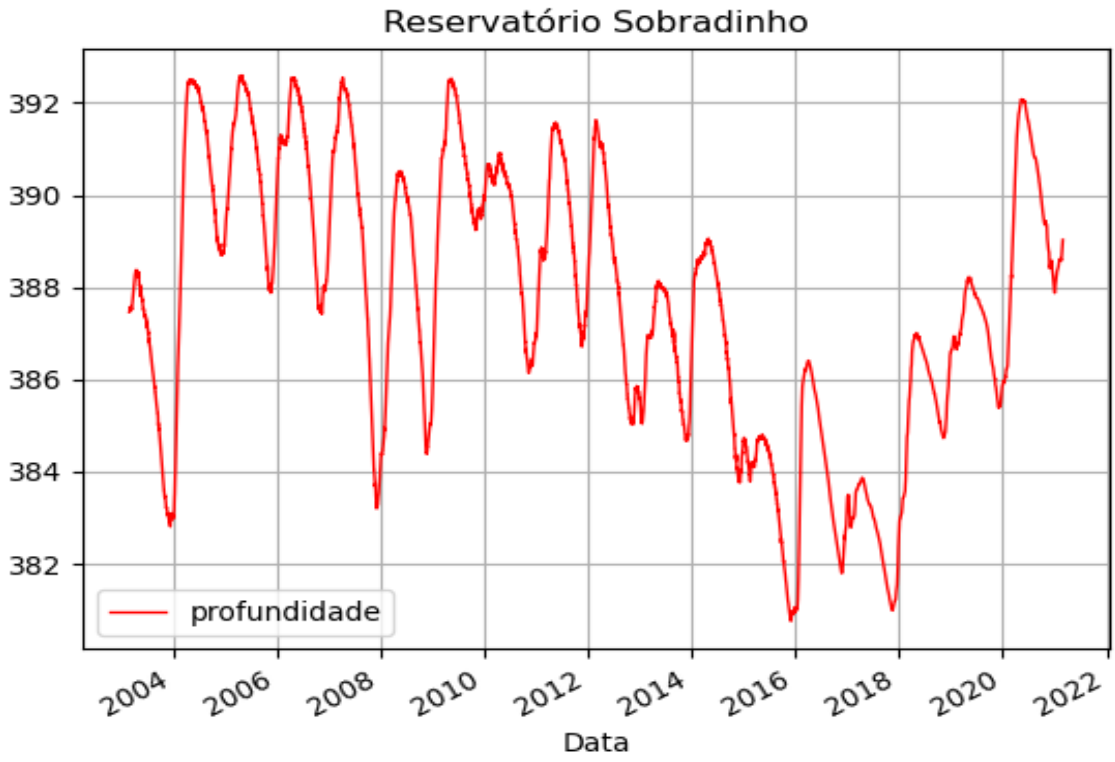

Figura – 5

Este gráfico demonstra o fator sazonalidade da série temporal, com variações bem acintosas de mais de 10 metros de profundidade no período entre 2013 até 2020, entretanto volta-se lentamente aos valores dos anos de 2004. A seca neste período e a volta a normalidade, demonstra que a região do reservatório de Sobradinho não é resiliente as mudanças climáticas.

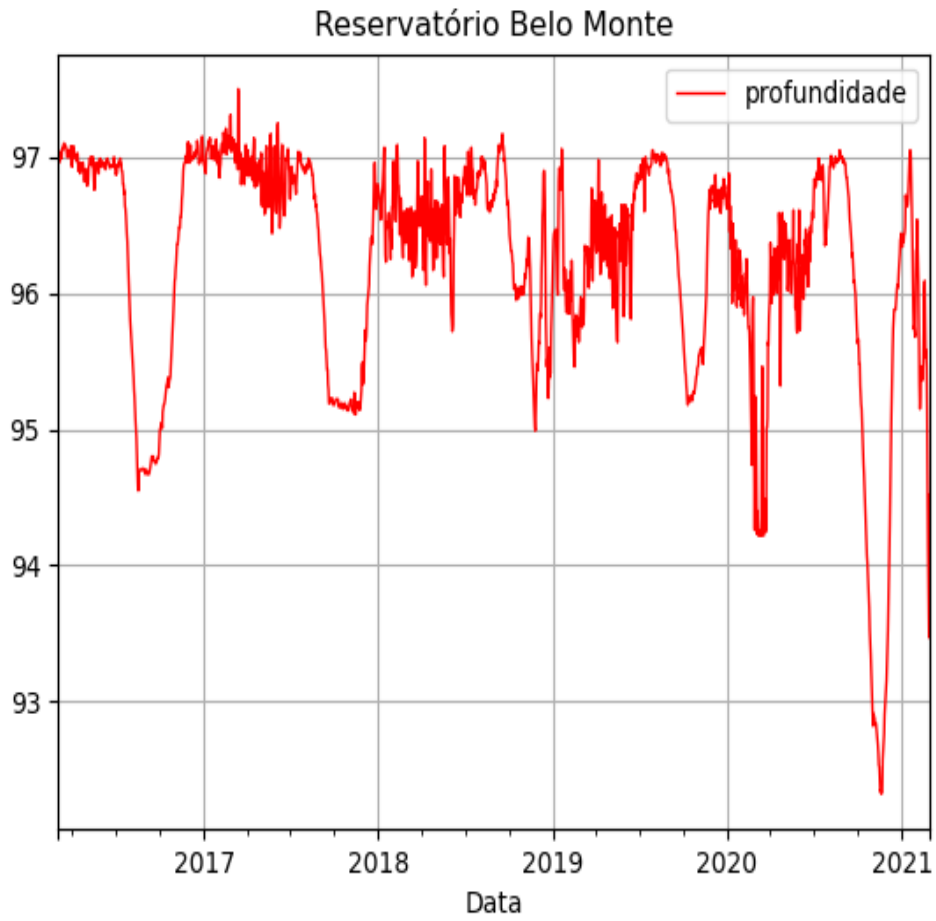

Figura – 6

Este reservatório é o mais recente dentre os escolhidos no trabalho, e demonstra grande variações provocadas por períodos curtos de dois à três meses provavelmente causado por intervenções de manutenção técnica. Assim os valores de baixa profundidade deveriam ser desprezados pela modelagem, já que não representam o comportamento real.

5. Criação de Modelos de Machine Learning

A criação dos modelos de aprendizado de máquina neste tópico, se faz utilizando as ferramentas Python/PyCharm configurado para utilizar o pacote Anaconda para gerenciar a configuração das bibliotecas que são importadas. Neste ambiente foca-se nos dados do reservatório de Sobradinho para demonstrar o roteiro de implementação dos modelos LSTM e ARIMA.

5.1 Modelo ARIMA (*Autoregressive Integrated Moving Average*)

ARIMA é um modelo de média móvel autorregressiva integrada, utilizada nas séries temporais para previsão. Ela requer que seja feita uma avaliação prévia de seu comportamento para obter parâmetros de chamada das suas funções.

5.1.1 Verificar se é uma série temporal estacionária

Uma série temporal é dita estacionária se as suas propriedades estatísticas, tais como a média e variância permanecem constantes ao longo do tempo.

Um dos métodos para identificar se a série temporal é estacionária é método ADF (*Augmented Dickey Fuller*), se ele retornar o valor p-value[1] abaixo de 0,05 fica comprovado que é estacionária. Na biblioteca statsmodels esta função não necessita de ajustes finos para obter um resultado confiável, mas é possível otimizar a função através de seus parâmetros como (e.g.: autolag:"AIC" ou "BIC").

```
from statsmodels.tsa.stattools import adfuller

print("p-value:", adfuller(dataFrame)[1])
```

Figura - 7

Como o resultado do p-value = 7.847096790972202e-05 foi bem inferior a 0,05 constata-se que a série deste reservatório é estacionária, como esperado pela avaliação do gráfico feita anteriormente.

Se neste teste a série não fosse dada como estacionária, utiliza-se o método de diferenciação para criar uma nova série até que o resultado do teste passe. A

biblioteca Pandas disponibiliza este recurso na forma: dataFrame.diff() para criar uma série diferencial D=1, dataFrame.diff().diff() para criar uma série diferencial D=2.

5.1.2 Obter os valores dos parâmetros p, d e q

O P representa o “modelo AR” do ARIMA, D o “I” e Q o “MA”, eles são os parâmetros mais importantes na chamada da função ARIMA na biblioteca *statsmodels.*

A forma convencional de obtê-los é a plotagem dos gráficos pirulitos “lollipop”: ACF para obter o parâmetro P que é a autocorrelação e o PACF para obter o parâmetro Q. O parâmetro D é o grau diferencial da série, mas como esta série é estacionária sem ter passado por método de diferenciação ele assume o valor D=0 (zero). Abaixo segue as funções para plotar os gráficos ACF e PACF e os gráficos gerados.

```
from matplotlib import pyplot plt as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(df.diff().diff().dropna(),lags=40)
plot_pacf(df.diff().diff().dropna(),lags=40)
plt.show()
```

Figura– 8

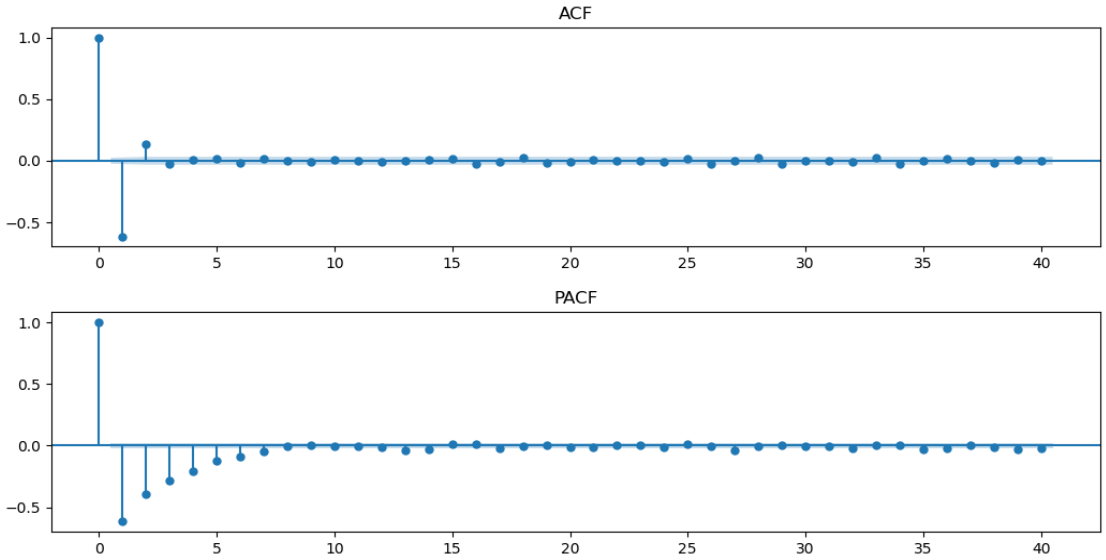

Figura - 9

Para obter o termo P:

O gráfico ACF tem uma mudança do primeiro *lollipop* do segundo?

a. Se não houver: P=0;

b. Se houver: olhe para o gráfico PACF e conte quantos *lollipops* estão fora antes de retornar para zona crítica, em nosso caso são 8 (oito). A zona crítica está marcada por uma faixa azulada em torno do eixo horizontal.

Para obter o termo Q:

O gráfico PACF tem uma mudança do primeiro *lollipop* do segundo?

a. Se não houver: Q=0;

b. Se houver: olhe para o gráfico ACF e conte quantos *lollipops* estão fora antes de retornar para zona crítica, em nosso caso são 1 (um).

5.1.3 Treinamento e previsão pelo modelo ARIMA

A biblioteca *statsmodels* permite que se faça divisão da massa de dados em parte para treinamento e outra para teste, assim pode-se comparar se o modelo convergiu corretamente, entretanto como a biblioteca também permite avaliar se houve falha no modelo através de um sumário estatístico, segue-se esta última abordagem. Abaixo segue a figura que demonstra as funções utilizadas neste processo.

```
arima = ARIMA(df, order=(8,0,1))

model_fit = arima.fit()
print(model_fit.summary())
prediction=model_fit.get_prediction(start='2021-03-01',end='2022-03-01')
predicted_values = prediction.predicted_mean
confidence_intervals = prediction.conf_int(alpha=0.05)
```

Figura - 10

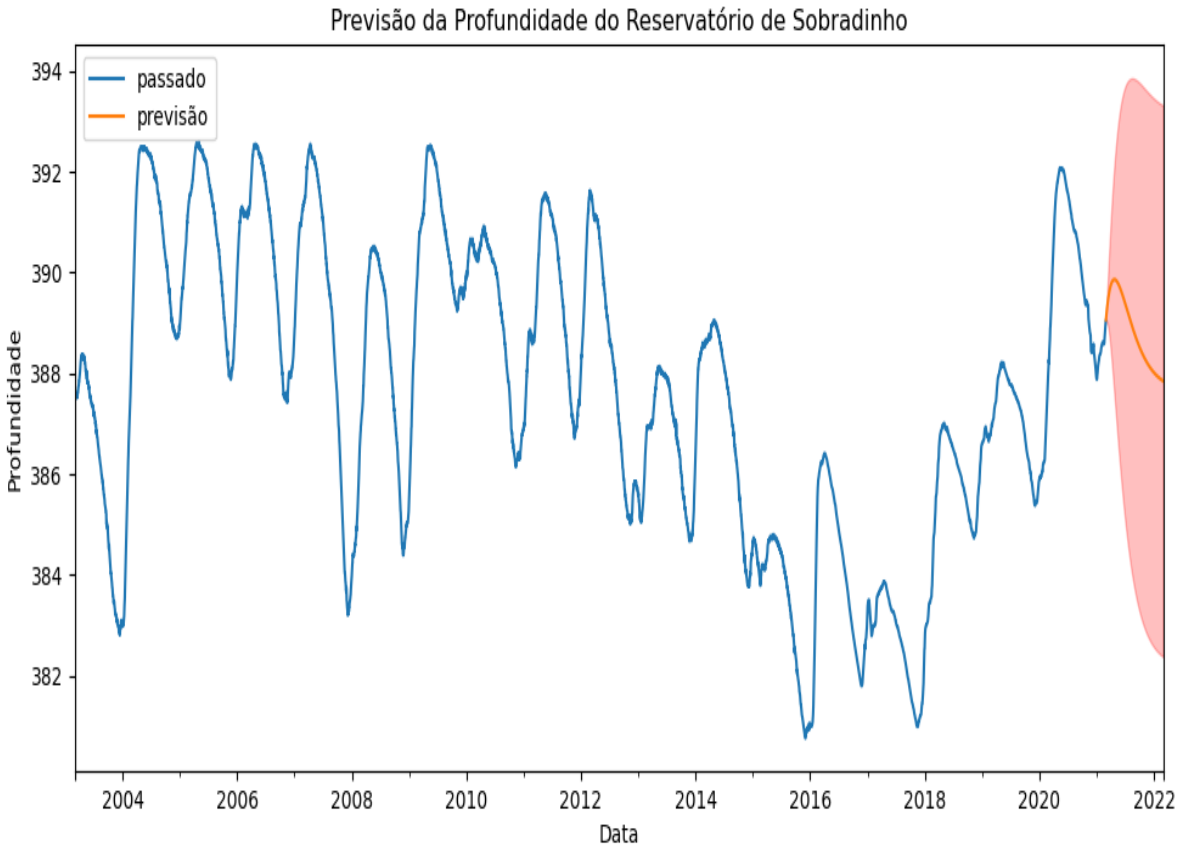

Figura - 11

5.1.4 Treinamento e previsão pelo modelo ARIMA Sazonal

O tratamento da sazonalidade na biblioteca ARIMA é feita através da introdução de novo parâmetro o seasonal_order=(P,D,Q,M), ele é obtido da mesma forma que o p,d,q anterior, somente que deve-se considerar o período em que ocorre o padrão da sazonalidade, e M é um valor que identifica se ela ocorre no dia, mês, ano com valores (e.g. 7, 12, 365).

A mudança no gráfico pode ser pequena assim, olhar o sumário e identificar se diminuiu os índices AIC, BIC e p>[z], são importantes para manter a escolha dos parâmetros.

5.2 Modelo RNN - LSTM (*Long Short Term Memory*)

A LSTM é uma variação das Redes Neurais Recorrentes (RNN), adequada para cenários de classificação e previsão de séries temporais, ela difere das redes neurais artificiais unidirecionais onde os sinais da entrada seguem direto para a saída sem feedback na mesma camada. Então a saída de uma camada não alimenta a mesma camada. As Redes Neurais Recorrentes (RNN) ou retroalimentadas são capazes de ter os sinais seguindo ambos os sentidos.

Para implementar esta arquitetura, utiliza-se a ferramenta Python/PyCharm, importando as bibliotecas *TensorFlow/Keras*, que contém o modelo sequencial do LSTM.

Para efeito de velocidade de desenvolvimento pode-se agrupar os dados dos reservatórios por mês ao invés de dias, reduzindo os tempos de processamento, mas em contrapartida o modelo fica ligeiramente mais errado.

5.2.1 Dividir a amostragem de dados obtida em treinamento e teste e normalizá-las.

Considerando que já se tem na variável new_data o *Dataframe* carregado com duas colunas: data e profundidade, pode-se iniciar o processamento, com a divisão da amostragem no grupo de treinamento e teste e depois normalizá-los para intervalos entre 0 e 1.

```
# GRUPO DE TREINAMENTO E TESTE
dataset = new_data.values
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# NORMALIZA DADOS
scaler = MinMaxScaler(feature_range=(0, 1))
scaled dataset = scaler.fit transform(dataset)
```

Figura - 12

5.2.2 Formatação dos dados para os parâmetros da função da biblioteca.

Uma das partes mais trabalhosas no modelo LSTM é a formatação dos parâmetros que são entregues para a função *fit()* do *TensorFlow/Keras;* a visualização destes parâmetros é importante para o entendimento da forma de processamento.

Inicialmente temos uma série temporal representada por data e valor, que deve gerar dois parâmetros para a função *fit()*, o TrainX e o TrainY.

O TrainY é um vetor simples contendo todos os valores da série, retirando os primeiros N valores que compõe a primeira janela, ou seja, o valor de *lookback* selecionado, neste caso com o valor 3. Desta forma usando a série abaixo, temos:

Série: 10, 11, 12, 13, 14, 15, 16

TrainY = [13, 14, 15, 16]

Já o TrainX é uma matrix 3-D, e para obtê-la temos duas etapas de transformação:

1) Criar uma matriz 2-D onde o número de colunas é o tamanho do *lookback* selecionado, ele é o parâmetro que define a janela de quantos valores passados são avaliados. Assim temos para série de exemplo acima com *loopback*=3, o seguinte TrainX:

TrainX = [[10, 11, 12], [11, 12, 13], [12, 13, 14], [13, 14, 15], [14, 15, 16]]

2) Dar um “reshape”, na matriz 2-D acima para virar 3-D, ficando assim.

TrainX = [[[10], [11], [12]], [[11], [12], [13]], [[12], [13], [14]], [[13], [14], [15]], [[14], [15], [16]]]

Assim o parâmetro *input_shape* na criação da camada LSTM, passa a ser uma matrix [3 x 1].

Desta forma pode-se entender que para cada valor de TrainY temos uma matriz 2-D [3x1] representando os valores passados. Assim temos TrainX como o domínio e TrainY como a Imagem.

5.2.3 Criação do modelo e previsão.

A biblioteca permite através da chamada *model.add()* modelar a quantidade de camadas, tipo de camada e número de neurônios, a função *model.compile()* define a otimização dos pesos internos e finalmente o *model.fit()* cria o modelo, que pode ser gravado em um arquivo no formato ".h5". O número de *Epochs* refere-se à quantidade de rodadas de treinamento que serão feitas com os mesmos dados, algo importante de ajuste quando se tem quantidades pequenas de amostras, neste caso *epochs*=10 para dados por dia e 100 para amostras agrupadas por mês.

```
# GRUPO DE TREINAMENTO E TESTE
dataset = new_data.values

model = Sequential()
model.add(LSTM(NUMERO_NEURONIOS, input_shape=(trainX.shape[1], 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=EPOCHS, batch_size=1, verbose=2)
```

Figura - 13

Para previsão chama-se a função "model.predict()" com os dados de teste, assim pode-se comparar seja pela impressão de um gráfico ou pelo cálculo do erro médio quadrádico chamando a função matth.sqrt(TesteY, testPredict), neste caso recupera-se os valores originais das profundidades antes.

```
testPredict = model.predict(testX)

testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

Figura - 14

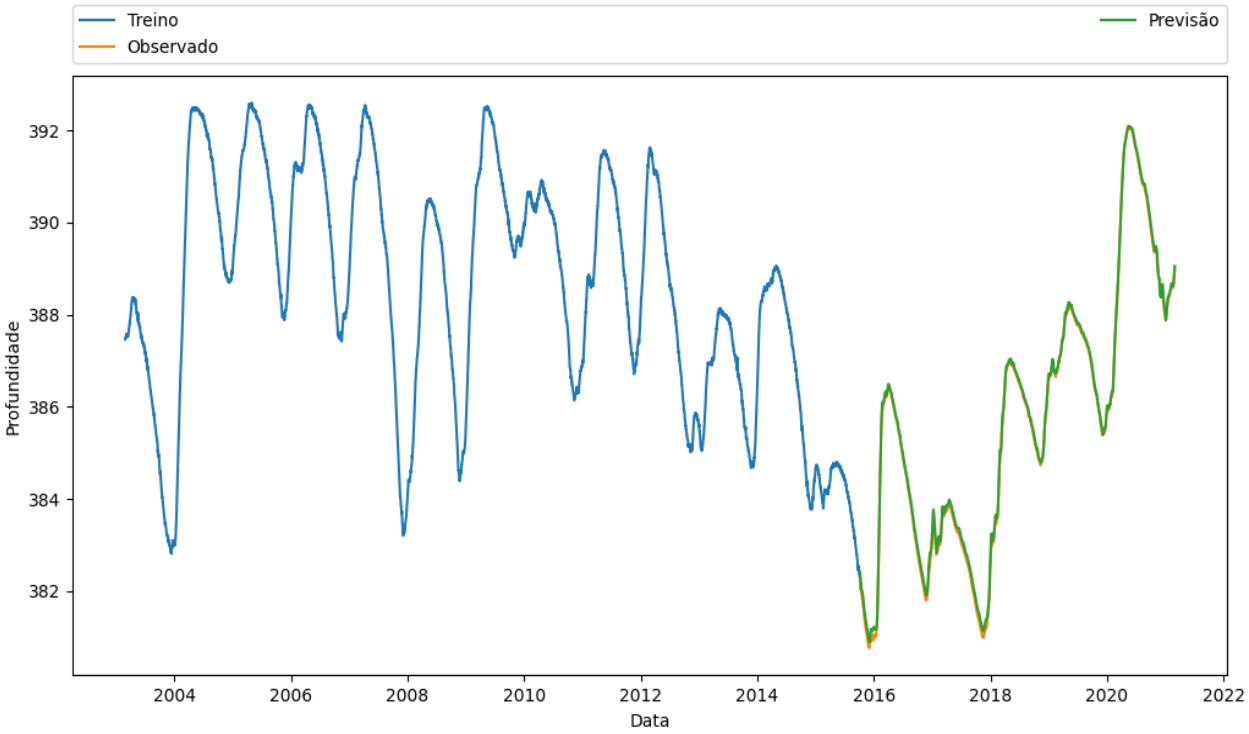

Figura – 15

6. Apresentação dos Resultados

O resultado dos testes em ambos os modelos LSTM e ARIMA, demonstrou que a previsão dos níveis dos reservatórios é totalmente viável mesmo quando se há fatores climáticos e não previsíveis. Neste caso a modelagem LSTM tende a ser melhor que o ARIMA que também apresentou bons resultados, mas a capacidade de esquecimento do LSTM torna-se importante para questões de imprevisibilidade.

A obtenção e preparação dos dados é simples, e como a coleta de dados é feita por instrumentos, dificilmente dados errados e fora da curva precisam ser eliminados, o processamento de 6500 registros mesmo na modelagem LSTM, levou seis minutos de processamento, para a maior configuração de processamento para um ótimo resultado de previsão.

Testes com variação do número de camadas no LSTM demonstrou que duas camadas são ótimas para o que foi exigido e para o ARIMA seguindo o modelo “pirulito” alcança-se facilmente os parâmetros P e Q, para ajuste do modelo. Quando o ajuste dos parâmetros não é bem-feito pode-se ver nos gráficos como do apêndice Figura – 16.

Fazer o agrupamento por mês dos valores aferidos de profundidade ao invés de dia como vem da fonte de dados, demonstrou uma piora da previsão, algo um tanto quanto inesperado, visto que reservatórios tendem a ter pequenas variações.

É visível que a modelagem por LSTM é mais trabalhosa, devido as transformações de matrizes para o formato das funções do modelo, enquanto o ARIMA e até a *Profet* que não está neste trabalho, são muito superiores no sentido facilidade de implementação.

Baseado nas previsões obtidas nos dois modelos, podemos deduzir que os reservatórios avaliados não estão e nem ficarão com níveis críticos para este ano, e que seguem o padrão de anos passados, mesmo que a quantidade de consumo tenha aumentado.

7. Links

Vídeo de apresentação: https://youtu.be/UeBUGFlyRQM
Repositório do Projeto: https://github.com/cloviszuben/PyCharm-LSTM-ARIMA
Repositório de Origem dos dados: http://www.ons.org.br

8. Apêndice

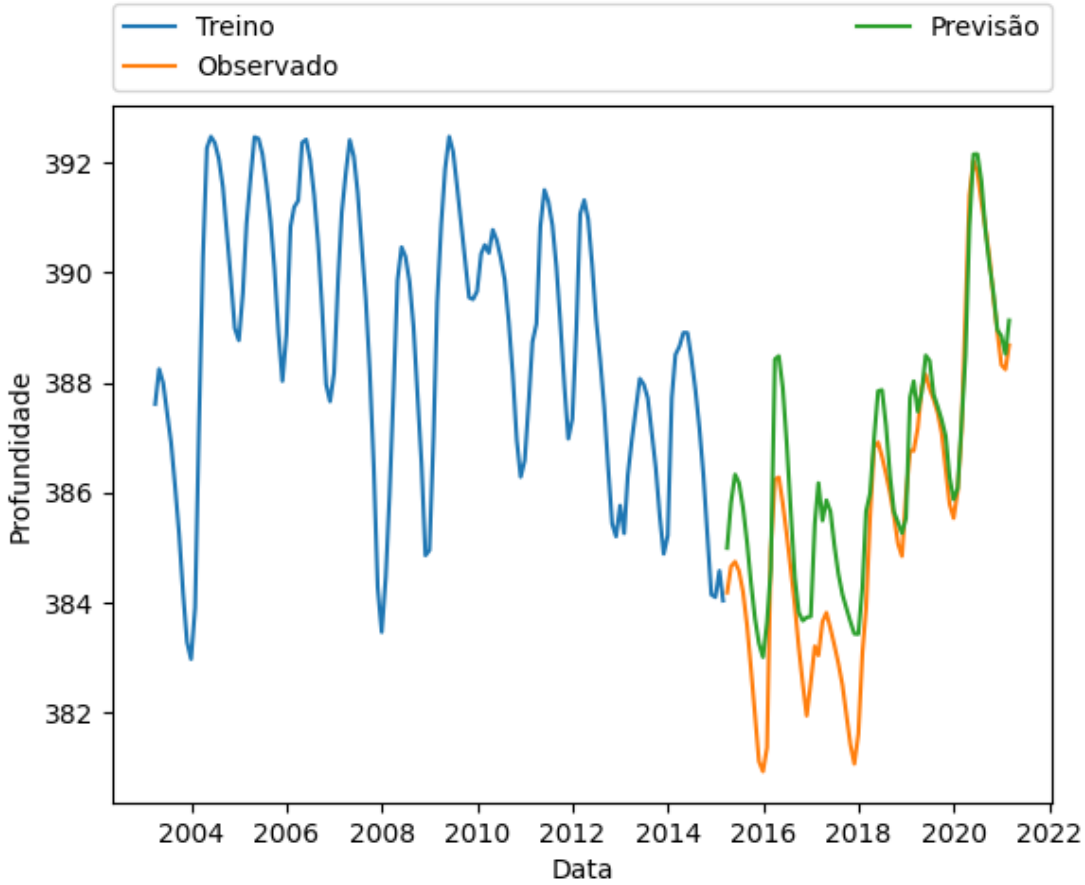

Figura – 16 (LSTM com ajuste malfeito)