Secretaria da Fazenda do Estado do Rio Grande do Sul
Tesouro do Estado
Divisão de Estudos Econômicos e Fiscais e Qualidade do Gasto

Redes Neurais Aplicadas na Previsão de Receita de ICMS no Rio Grande do Sul

Gabriel Zuanazzi Dornelles[1]
Fernando Roberto Schwartzer[2]
Jacó Braatz[3]

TEXTOS PARA DISCUSSÃO TE/RS Nº 19

janeiro/2022

Publicação cujo objetivo é divulgar resultados de estudos direta ou indiretamente desenvolvidos pelo Tesouro do Estado, ou de interesse da instituição, os quais, por sua relevância, levam informações para profissionais especializados e estabelecem um espaço para sugestões e debates de ideias. Todas as contribuições recebidas passam, necessariamente, por avaliação de admissibilidade e por análise dos pares. As opiniões emitidas nesta publicação são de exclusiva e inteira responsabilidade do(s) autor(es), não exprimindo, necessariamente, o ponto de vista do órgão.

[1] Auditor Fiscal da Receita Estadual, especialista em projetos. E-mail: gabrielzd@sefaz.rs.gov.br
[2] Arquiteto da Receita Municipal, Cientista de Dados. E-mail: fernando.schwartzer@portoalegre.rs.gov.br
[3] Auditor Fiscal da Receita Estadual, Doutor em Economia. E-mail: jacob@sefaz.rs.gov.br

Resumo

Neste artigo desenvolveu-se um modelo univariado de curto prazo utilizando Redes Neurais de Memória de Longo e Curto Prazo (LSTM) com intuito de prever a arrecadação mensal do Imposto de Circulação de Mercadorias e Serviços do Estado do Rio Grande do Sul. Conclui-se que o modelo utilizando redes neurais apresentou erro de previsão acumulado de -2,33% em seis previsões de um passo a frente (T+1), apresentando ganhos significativos frente a outros métodos preditivos já utilizados pela SEFAZ-RS.

1. Introdução

Previsão de séries temporais compõe um campo de pesquisa tão rico quanto desafiador. Previsões de condições climáticas, de mercado, de vendas e até mesmo de propagação de vírus em tempos de pandemia são alguns exemplos das tarefas impostas diariamente aos profissionais desta área. Tais desafios motivaram inúmeros cientistas a buscarem o "melhor modelo", a abordagem otimizada para cada problema. Entretanto, por óbvio, esta resposta não é universal, sequer atemporal, devendo ser buscada para cada campo de pesquisa, para cada base de dados estudada.

Nesta lógica, a gestão do Estado do Rio Grande do Sul é fortemente influenciada por previsões de séries temporais. A Constituição Brasileira, bem como a Lei de Responsabilidade Fiscal (LRF) e outros regramentos infraconstitucionais, determinam que os entes federativos realizem o adequado planejamento orçamentário, publicizados através de leis orçamentárias. Os diferentes instrumentos legais, que materializam este orçamento público, possuem um ponto de convergência: as despesas futuras são fixadas enquanto as receitas futuras são estimadas. Justamente neste último ponto, a estimativa de receitas, que o desenvolvimento e aplicação de modelos cada vez mais robustos se faz necessário na perspectiva do Tesouro do Estado do Rio Grande do Sul.

As estimativas de receitas supracitadas influenciam diretamente na saúde fiscal do Estado. A partir desta definição, são calculadas, por exemplo, a meta de resultado primário do ano, o resultado orçamentário e a variação da dívida prevista. Ademais, este impacto na execução orçamentária desdobra na necessidade ou não de contingenciamento de recursos financeiros por parte dos Poderes e Órgãos com autonomia financeira. O próprio estabelecimento do orçamento de todos os órgãos da administração pública, bem como os mínimos constitucionais para saúde e educação, depende desta previsão, o que, em última instância, determinará a qualidade do gasto final em favor do cidadão.

Por este fato, a própria LRF determina que as previsões de arrecadação devem atender um rigor técnico, constituindo requisitos essenciais da responsabilidade fiscal:

Art. 11. Constituem requisitos essenciais da responsabilidade na gestão fiscal a instituição, previsão e efetiva arrecadação de todos os tributos da competência constitucional do ente da Federação.

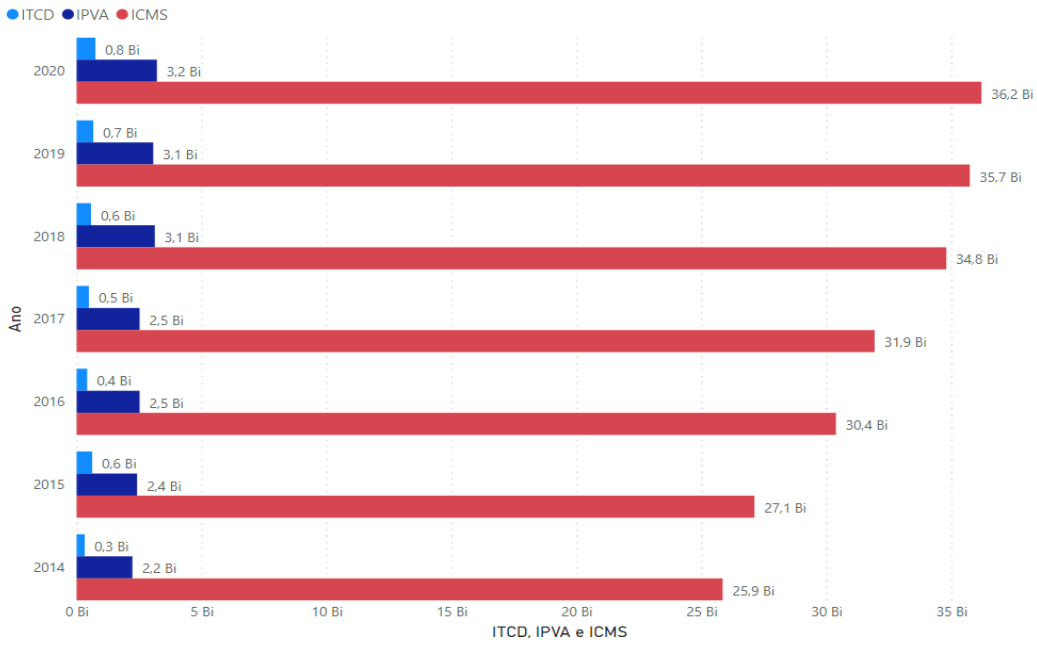
...

Art. 12. As previsões de receita observarão as normas técnicas e legais, considerarão os efeitos das alterações na legislação, da variação do índice de preços, do crescimento econômico ou de qualquer outro fator relevante e serão acompanhadas de demonstrativo de sua evolução nos últimos três anos, da projeção para os dois seguintes àquele a que se referirem, e da metodologia de cálculo e premissas utilizadas.

De acordo com o Estudo do Banco Central Europeu (LEAL, 2007), embora a área de previsões fiscais seja vista por muitos predominantemente como uma arte do que pura ciência, é consenso que esta deve ser lastreada por métodos transparentes e procedimentos claros. Somente deste modo agentes políticos, participantes do mercado e o público em geral podem ter uma informação precisa sobre a evolução orçamentária. Tais previsões são a principal ferramenta para os formuladores de políticas tomarem decisões fiscais adequadas, para o público avaliar a necessidade de tais decisões e, posteriormente, avaliar o sucesso de suas implementações. Destarte, este estudo destaca a necessidade de obtenção de técnicas de previsões econômicas e fiscais cada vez mais consistentes e acuradas, com ênfase na transparência metodológica.

A partir do supracitado, este trabalho tem como objetivo elaborar um modelo de previsão da arrecadação do ICMS (Imposto sobre Circulação de Mercadorias e Prestação de Serviços de Transporte Interestadual e Intermunicipal e de Comunicação) utilizando uma estrutura de Redes Neurais de Memória de Longo e Curto Prazo (LSTM). Em seguida, compara-se os resultados obtidos com efetiva arrecadação do período, bem como com os resultados gerados por previsões já utilizadas na SEFAZ/RS. A escolha por utilizar a arrecadação do ICMS como objeto do modelo deve-se ao fato de o mesmo ser o principal imposto estadual na perspectiva de seu valor representativo, sendo responsável por aproximadamente 90,6% do total arrecadado de impostos estaduais, conforme demonstrado pela figura 01.

Figura 01: Distribuição da arrecadação dos impostos estaduais RS



Fonte: elaboração autores

Este trabalho está estruturado da seguinte maneira: na primeira parte do trabalho é apresentada a relevância da acurácia de tais previsões e a importância de métodos transparentes/incrementais, os quais influenciam – em última instância – na própria qualidade do gasto público. Em seguida, no segundo e terceiro capítulo, expõe-se uma breve fundamentação teórica e exposição de motivos pela escolha específica das redes LSTM, assim como exemplos de trabalhos relacionados bem-sucedidos nesta área. Na quarta parte, detalha-se o algoritmo proposto desde o pré-processamento dos dados, definição/otimização dos Hiper Parâmetros, treinamento do modelo e, por fim, previsão fora da amostra. Outrossim, com intuito de posicionar tal publicação como um ponto de partida e não um trabalho com fim em si, garantindo a devida transparência e possibilidade de aprimoramentos por terceiros, disponibilizou-se o código aberto desenvolvido pelos autores no Anexo A

2. Metodologia

Esta seção está dividida em cinco subseções. Na primeira parte define-se o que é uma Rede Neural Aritifical (RNA) e seu funcionamento. No item 2.2 apresenta-se um tipo de RNA, as Redes Neurais Recorrentes (RNN), as quais possuem aptidão para tratar problemas sequenciais. Posteriormente, no item 2.3, aborda-se as Redes Neurais de Memória de Longo e Curto Prazo (LSTM), modelo escolhido para este trabalho. Por fim, no item 2.4 apresenta-se o método para escolher a melhor arquitetura para o modelo.

2.1 Rede Neural Artificial

Uma rede neural artificial (RNA) é a parte de um sistema de computação projetada para simular a maneira como o cérebro humano analisa e processa informações. É a base da inteligência artificial (IA) e resolve problemas que seriam impossíveis ou difíceis para os padrões humanos ou estatísticos. As RNAs têm recursos de autoaprendizagem que lhes permitem produzir melhores resultados à medida que mais dados se tornam disponíveis.

Uma RNA possui centenas ou milhares de neurônios artificiais chamados unidades de processamento, que são interconectados por nós. Essas unidades de processamento são compostas por unidades de entrada e saída. As unidades de entrada recebem várias formas e estruturas de informação com base em um sistema de ponderação interno, sendo que a rede neural tenta aprender sobre as informações apresentadas para produzir um relatório de saída. Assim como os humanos precisam de regras e diretrizes para chegar a um resultado ou saída, as RNAs também usam um conjunto de regras de aprendizado chamado retropropagação (backpropagation), uma abreviatura para propagação reversa de erro, para aperfeiçoar seus resultados de saída.

Uma RNA inicialmente passa por uma fase de treinamento, onde aprende a reconhecer padrões em dados, seja visual, auditiva ou textualmente. Durante essa fase supervisionada, a rede compara a saída real produzida com o que deveria produzir - a saída desejada. A diferença entre os dois resultados é ajustada usando retropropagação (backpropagation). Isso significa que a rede trabalha para trás, indo da unidade de saída para as unidades de entrada para ajustar o peso de suas conexões entre as unidades até que a diferença entre o resultado real e o desejado produza o menor erro possível.

2.2 Redes Neural Recorrente

Uma rede neural recorrente (RNN) é um tipo de rede neural artificial que usa dados sequenciais ou dados de série temporal. Esses algoritmos de aprendizado profundo são comumente usados para problemas ordinais ou temporais, como tradução de linguagem, processamento de linguagem natural (NLP), reconhecimento de fala e legendagem de imagens; eles são incorporados a aplicativos populares, como Siri, pesquisa por voz e Google Translate.

Como as redes neurais de feedforward e convolucionais (CNNs), as redes neurais recorrentes utilizam dados de treinamento para aprender. Elas se distinguem por sua "memória", pois recebem informações de entradas anteriores para influenciar a entrada e a saída atuais.

Enquanto as redes neurais profundas tradicionais presumem que as entradas e saídas são independentes umas das outras, a saída das redes neurais recorrentes depende dos elementos anteriores dentro da sequência. Embora eventos futuros também sejam úteis para determinar a saída de uma determinada sequência, as redes neurais recorrentes unidirecionais não conseguem levar em conta esses eventos em suas previsões.

2.3 Rede *Long Short Term Memory*

Redes Neurais de Memória de Longo e Curto Prazo (LSTM) são um tipo de Rede Neural Recorrente em Aprendizado Profundo, apresentado por Hochreiter e Schmiduber em 1997, sendo desenvolvido especificamente para o uso do tratamento de problemas de predição sequencial. Por exemplo: previsão do tempo, previsão do mercado de ações, recomendação de produto, geração de texto / imagem / manuscrito e tradução de textos.

Como outras redes neurais, elas contêm neurônios para realizar cálculos; no entanto, para as LSTM, eles são frequentemente chamados de células de memória ou simplesmente células. Essas células contêm pesos e portas; as portas são a característica distintiva dos modelos LSTM. Existem 3 portas dentro de cada célula. A porta de entrada, a porta de esquecimento e a porta de saída.

Os modelos LSTM são capazes de olhar para trás em dados e decisões anteriores e tomar decisões a partir deles. Também usam o mesmo processo para fazer suposições / previsões fundamentadas sobre o que pode acontecer. Portanto, sendo alimentado com

dados sequenciais, encontrará tendências e as usará para prever os resultados futuros. A metodologia utilizada nesse trabalho envolve modelos de Redes Neurais Recorrentes do tipo LSTM devido à eficácia histórica de seus resultados em relação à previsão de séries temporais não lineares e a exploração e extração de informações que pode ser obtida a partir dos dados de entrada.

A estrutura é formada por uma camada LSTM, seguida por uma camada Dropout, outra camada LSTM com o mesmo número de unidades da anterior, outra camada Dropout, finalizando com uma camada Dense. A métrica utilizada foi o Erro Quadrático Médio (MSE), computando a distância quadrada entre as entradas, retornando o valor médio sobre a última dimensão.

A arquitetura de rede neural com duas camadas LSTM visou um modelo com maior poder de representação. Esse maior poder de representação significa que a rede neural pode caber em funções mais complexas e generalizar bem os dados de treinamento. A desvantagem de utilizar essas redes neurais mais profundas é que elas são altamente propensas ao sobreajuste (overfitting).

O sobreajuste (overfitting) é um problema comum que é definido como a incapacidade de um modelo de aprendizado de máquina treinado generalizar bem para dados invisíveis, mas o mesmo modelo tem um bom desempenho nos dados em que foi treinado. Por isso, visando minimizar os efeitos do sobreajuste (overfitting), foram utilizadas camadas Dropout. Deste modo, estas camadas funcionam reduzindo aleatoriamente o número de neurônios interconectados em uma rede neural. A cada etapa do treinamento, cada neurônio tem uma chance de ser desconsiderado, ou melhor, excluído (dropped out) da contribuição agrupada dos neurônios conectados. Tal fato minimiza o sobreajuste (overfitting) ao passo que cada neurônio se torna independentemente suficiente, no sentido de que os neurônios dentro das camadas aprendem valores de peso que não são baseados na cooperação de seus neurônios vizinhos. Consequentemente, é reduzida a dependência de um grande número de neurônios interconectados para gerar um poder de representação adequado da rede neural treinada.

Por fim, foi utilizada uma camada Dense, que é uma camada de rede neural profundamente conectada, o que significa que cada neurônio na camada Dense recebe entrada de todos os neurônios de sua camada anterior.

2.4 Análise e Avaliação

As redes passam por ciclos de treinamento e validação, que consistem em atualizar os pesos da rede, a partir do erro observado com as previsões dos dados de treinamento, e em seguida avaliar o desempenho do modelo treinado por meio do erro médio quadrático (do inglês Mean Squared Error - MSE).

$$\textbf{MSE} = \frac{\sum_{i=1}^{n} (\text{real} - \text{previsto})^2}{n}$$

Para cada topologia avaliada, as etapas de treinamento e validação foram realizadas consecutivamente e múltiplas vezes executadas. Importante salientar que os intervalos testados dos Hiper Parâmetros utilizados nesta avaliação foram determinados a partir de uma grande amplitude entre o menor e o maior valor, convergindo para as faixas de valores que apresentaram os menores erros.

Dessa forma, para escolher a melhor arquitetura para o modelo de redes neurais utilizados neste trabalho (LSTM) foi calculado o erro médio de validação utilizando o erro médio quadrático (MSE), com diferentes configurações dos parâmetros:

- Units: número de unidades ocultas ou células ocultas dentro da camada LSTM;

- Dropout; um método de regularização para reduzir o overfitting e melhorando a performance do modelo.

- Batch Size (tamanho do lote): limita o número de amostras a serem mostradas à rede antes que uma atualização de peso possa ser realizada. Essa mesma limitação é então imposta ao fazer previsões com o modelo de ajuste.

3. Trabalhos Relacionados

Analisar a relação oculta entre os dados fiscais históricos e usar modelos matemáticos para prever a receita fiscal futura é o foco desta pesquisa de previsão fiscal, sendo os modelos LSTM adequados para o problema.

Nesse sentido, Silva, H., Figueiredo, T. (2020) fizeram um trabalho de uso de modelos mais novos e acurados de Machine Learning, como Long Short-Term Memory (LSTM), para prever a receita de tributos do Rio de Janeiro. Os resultados apresentam erro relativo menor que 1%, para previsão do ICMS, indicando desempenho superior às previsões realizadas pela SEFAZ-RJ e aos modelos MLP, usados para efeitos de comparação.

S. Siami-Namini, N. Tavakoli and A. S. Namin (2019) relataram que as Redes Neurais Recorrentes (RNN) artificiais com memória, como a Memória de Longo Prazo (LSTM), apresentaram resultados superiores em comparação com a Média Móvel Integrada Autoregressiva (ARIMA) com uma grande margem. Os modelos baseados em LSTM incorporam "portas" adicionais com o propósito de memorizar sequências mais longas de dados de entrada.

Segundo Van Houdt, G., Mosquera, C. & Nápoles, G. (2020), as redes neurais recorrentes mostraram um desempenho bastante discreto até o LSTM aparecer. Uma razão para o sucesso dessa rede recorrente está em sua capacidade de lidar com o problema do gradiente de explosão / desaparecimento, que é uma questão difícil de ser contornada ao treinar redes neurais recorrentes ou muito profundas.

4. Aplicação Empírica

Com o intuito de avaliar criticamente os resultados obtidos através destas Redes Neurais Recorrentes do tipo LSTM, o experimento proposto foi projetado para realizar previsões iterativas de curto prazo. Curto prazo, pois, a previsão do modelo refere-se ao período de um mês no futuro (T+1) e iterativo porque após tal medição, através de um loop, o algoritmo compara o valor calculado com a efetiva arrecadação deste mês, registra este erro, substitui o valor calculado pela arrecadação real e executa novamente outra previsão para o segundo mês do intervalo (T+2). O intervalo selecionado para o estudo compreende 8 meses de previsão, sempre em (T+1): janeiro de 2021 a agosto de 2021. Portanto, para cada previsão em (T+1) foram utilizadas subamostras diferentes. Para a previsão de janeiro de 2021 foi utilizada uma subamostra com dados até dezembro de 2020 para a estimação. Na sequência, para a previsão de fevereiro de 2021, o modelo foi rodado novamente, sendo utilizada uma subamostra com dados até janeiro de 2021. Para

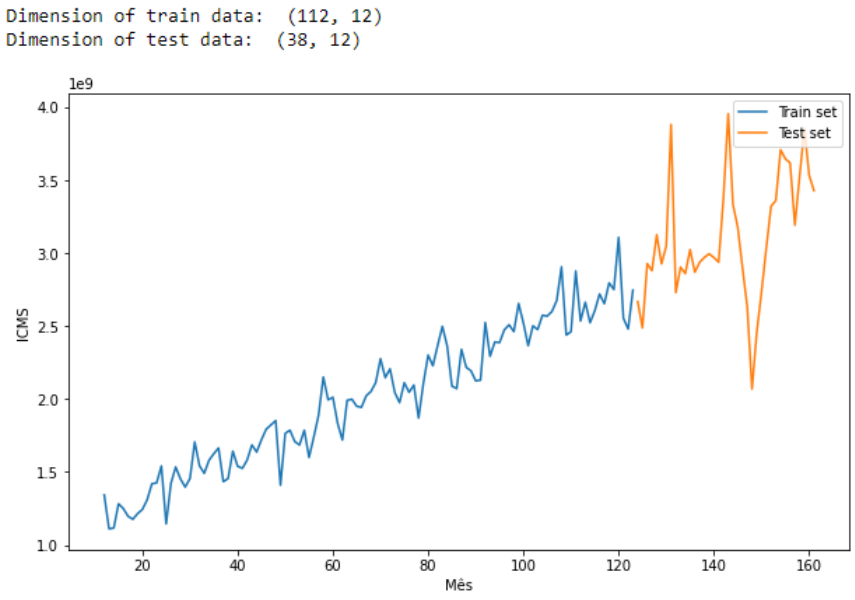
a previsão de março de 2021, foi utilizada outra subamostra com dados até fevereiro e assim sucessivamente. Em outras palavras, para cada previsão, o modelo foi rodado novamente.

4.1 Pré Processamento dos Dados

A série temporal usada neste trabalho foi extraída dos dados de arrecadação mensal do tributo ICMS do Estado do Rio Grande do Sul, presente no site da SEFAZ-RS, sendo composta por valores nominais de arrecadação em reais (R$) a partir de janeiro de 2009.

A base de dados foi dividida em uma base de treino (train test), com os dados 75% iniciais, e uma base de testes (test set) com os 25% restantes, mais recentes, conforme demonstrada na figura 02. Os dados de data foram tratados com uma função da biblioteca Datetime, de linguagem Python, que retorna o ordinal gregoriano de uma data.

Figura 02: Separação da base utilizada em treino e teste.



Fonte: elaboração autores

Tanto a base de treino quanto a base de testes foram dimensionadas com a ferramenta de pré-processamento de dados MinMaxScaler da biblioteca Scikit-learn (originalmente scikits.learn), que é uma biblioteca de aprendizado de máquina de código

aberto para a linguagem de programação Python. Inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, gradient boosting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy. O dimensionamento consistiu na tradução de cada característica individualmente de modo que esteja na faixa dada no conjunto de treinamento entre zero e um, ou seja, normalização dos dados conforme descrito na figura 03.

Figura 03: Normalização da base de dados.

```
In [14]: from sklearn.preprocessing import MinMaxScaler

In [15]: scaler_x = MinMaxScaler(feature_range = (0,1))
         scaler_y = MinMaxScaler(feature_range = (0,1))

In [16]: input_scaler = scaler_x.fit(X_train)
         output_scaler = scaler_y.fit(y_train)

In [17]: train_y_norm = output_scaler.transform(y_train)
         train_x_norm = input_scaler.transform(X_train)

In [18]: test_y_norm = output_scaler.transform(y_test)
         test_x_norm = input_scaler.transform(X_test)
```

Fonte: elaboração autores

Nos dados também foi aplicado um redimensionamento das matrizes para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A primeira dimensão representa o tamanho do lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

Figura 4: Redimensionamento da matriz.

```
In [19]: print('train_y_norm.shape: ', train_y_norm.shape)
         print('train_x_norm.shape: ', train_x_norm.shape)
         print('test_y_norm.shape: ', test_y_norm.shape)
         print('test_x_norm.shape: ', test_x_norm.shape)

         train_y_norm.shape:  (112, 1)
         train_x_norm.shape:  (112, 11)
         test_y_norm.shape:  (38, 1)
         test_x_norm.shape:  (38, 11)

         Redimensionamento da matriz para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A
         primeira dimensão representa o tamanho do lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma
         sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

In [20]: X_test = test_x_norm.reshape(38, 1, 11)
         X_train = train_x_norm.reshape(112, 1, 11)
         y_test = test_y_norm.reshape(38,1)
         y_train = train_y_norm.reshape(112, 1)
```

Fonte: elaboração autores

A partir do conjunto de dados de entrada, foram criadas novas variáveis em decorrência de manipulações matemáticas, conforme realizado nos trabalhos [Ticona 2013] e [Silva et al 2020]. Tais manipulações consistem em médias móveis e lag's, que são retardos nos valores originais da série. Sendo assim, nesse conjunto são criadas 11 variáveis, conforme pode ser observado na Tabela 1:

Tabela 01 - Cálculos realizados sobre os valores da série.

Número da Variável	Variável de Entrada	Cálculos Realizados Sobre os Valores da Série (V)
1	MM12	$Vn{-}1 + Vn{-}2 + \cdots + Vn{-}11 + Vn{-}12$ 12
2	MM6	$Vn{-}1 + Vn{-}2 + \cdots + Vn{-}5 + Vn{-}6$ 6
3	MM3	$Vn{-}1 + Vn{-}2 + Vn{-}3$ 3
4	MM2	$Vn{-}1 + Vn{-}2$ 2
5	L12	$Vn{-}12$
6	L6	$Vn{-}6$
7	L4	$Vn{-}4$
8	L3	$Vn{-}3$
9	L2	$Vn{-}2$
10	L1	$Vn{-}1$
11	rótulo	Vn=13

Fonte: Silva, H., Figueiredo, T. (2020)

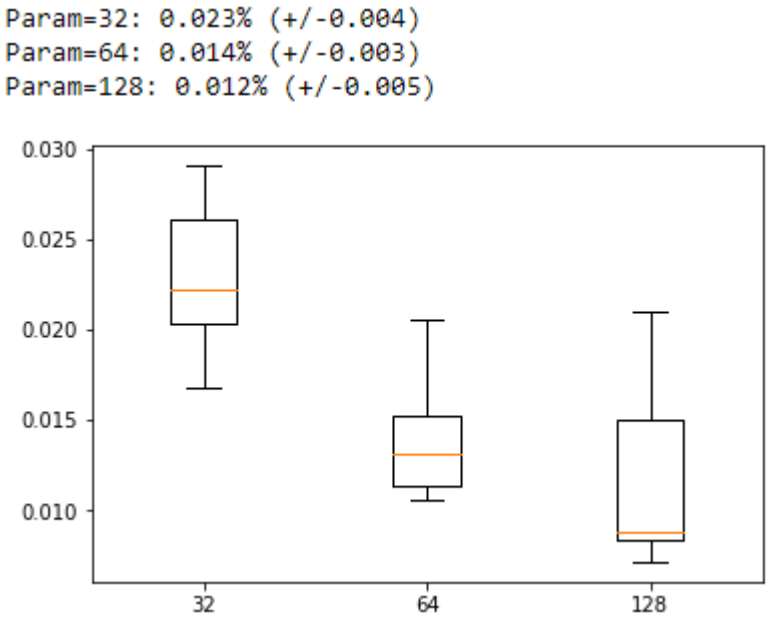
As quatro primeiras variáveis são as médias móveis, dos últimos 12, seis, três e dois meses, respectivamente. As variáveis numeradas de 5 a 10 são estabelecidas a partir de lag's em relação ao mês de saída, e a última é a saída (rótulo) com o valor da receita para o mês a ser previsto.

4.2 Definição/Otimização dos Hiper Parâmetros

Conforme (Reimers, 2017), a seleção otimizada dos parâmetros a serem utilizados para uma arquitetura de redes neurais representa, muitas vezes, a diferença entre redes medíocres para redes altamente eficazes. É consenso que esta definição representa importante papel no resultado final. Entretanto, tal área apresenta pouca literatura publicada para avaliar cientificamente seus reais impactos. Snoek et al. (2012) escreve que a otimização de hiperparâmetros "*... is often a black art that requires expert experience, unwritten rules of thumb, or sometimes brute-force search.*".

Desta maneira, com intuito de diminuir a arbitrariedade supracitada, normalmente utilizada nestas definições, e trazer uma análise científica para o presente trabalho, determinou-se para cada hiperparâmetro uma série de valores possíveis. Em seguida, estes foram avaliados através de processos iterativos para determinar os mais adequados para a base de dados utilizada, conforme figuras abaixo.

Figura 5: Iteração para definir no número de neurônios (*units*).

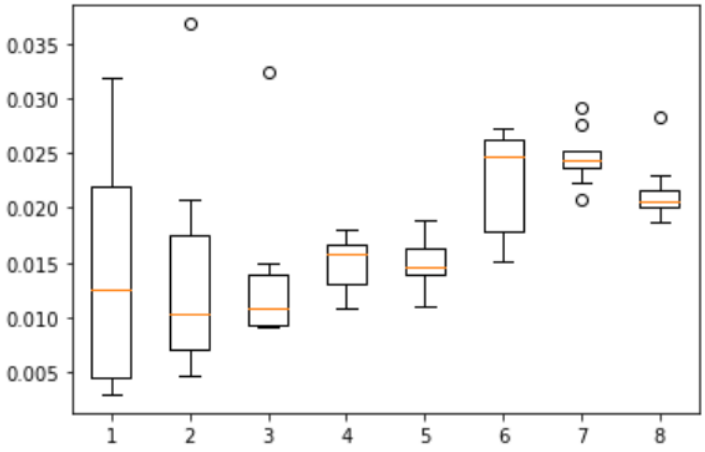


Fonte: elaboração autores

Nesta primeira otimização de hiperparâmetro é possível perceber que o número de neurônios (units) com o menor erro está entre 64 e 128 unidades. O experimento com 128 unidades, apesar de resultar na menor mediana para o erro, apresentou a maior dispersão entre os 3 valores para o hiperparâmetro testado, indicando a possibilidade de sobreajuste. Por isso, foram realizados posteriores experimentos para se aproximar do número de unidades que resultam no menor erro e menor dispersão dos resultados. Após este segundo teste, a estrutura com 80 unidades apresentou o melhor resultado.

Figura 6: Iteração para definir o Tamanho do Lote (*batchsize*).

```
Param=1: 0.014% (+/-0.010)
Param=2: 0.014% (+/-0.009)
Param=3: 0.013% (+/-0.007)
Param=4: 0.015% (+/-0.002)
Param=5: 0.015% (+/-0.002)
Param=6: 0.023% (+/-0.005)
Param=7: 0.025% (+/-0.002)
Param=8: 0.021% (+/-0.003)
```

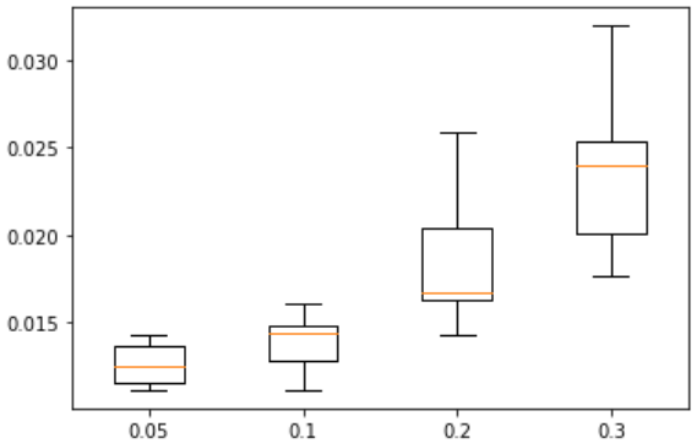


Fonte: elaboração autores

A Figura 6 apresenta o teste para a identificação do melhor Tamanho do Lote (batchsize). Verificou-se que o tamanho 5, apesar de não ter apresentado a menor mediana para o erro, resultou numa baixa dispersão dos erros e uma distribuição com maior simetria.

Figura 7: Iteração para definir o modelo com diluição (Dropout).

```
Param=0: 0.013% (+/-0.001)
Param=0: 0.014% (+/-0.001)
Param=0: 0.018% (+/-0.004)
Param=0: 0.023% (+/-0.004)
```



Fonte: elaboração autores

As iterações para definir o valor ótimo para o hiperparâmetro diluição (Dropout) demonstraram que a não utilização de 5% (0,05) das conexões de entrada para a ativação e atualizações de peso durante o treinamento apresentou o melhor desempenho para se evitar o sobreajuste e se obter o menor erro.

4.3 Treinamento

Um modelo de treinamento é um conjunto de dados usado para treinar um algoritmo de Aprendizado de Máquina. Consiste nos dados de saída de amostra e nos conjuntos correspondentes de dados de entrada que têm influência na saída. O modelo de treinamento é usado para executar os dados de entrada por meio do algoritmo para correlacionar a saída processada com a saída de amostra. O resultado dessa correlação é usado para modificar o modelo.

Este processo iterativo é denominado “model fitting” (ajuste do modelo). A precisão do conjunto de dados de treinamento ou do conjunto de dados de validação é crítica para a precisão do modelo.

O treinamento de modelo em linguagem de máquina é o processo de alimentar um algoritmo de Aprendizado de Máquina com dados para ajudar a identificar e aprender bons valores para todos os atributos envolvidos.

O aprendizado supervisionado, utilizado neste trabalho, é possível quando os dados de treinamento contêm os valores de entrada (as médias móveis e lags utilizadas neste trabalho, por exemplo) e saída (a arrecadação do ICMS, neste trabalho). Cada conjunto de dados que possui as entradas e a saída esperada é chamado de sinal de supervisão. O treinamento é feito com base no desvio do resultado processado do resultado documentado quando as entradas são alimentadas no modelo.

Em outras palavras, com intuito de trazer uma explicação menos técnica, todavia mais acessível, este treinamento consiste em uma fase em que são apresentados tanto os inputs e quanto os outputs para o algoritmo, ou seja, a máquina sabe tanto o resultado de entrada como o que deveria ser o resultado final. Em virtude disso, é possível que a máquina execute seus cálculos independentes, mas possa – ao final deste processo – comparar com o que deveria ser o resultado correto, calculando seu desvio e, por consequência, os ajustes necessários em sua estrutura matemática. O modelo final foi

treinado com camadas LSTM de 80 unidades, dropout de 0.05 e batch size 5, conforme resumo a seguir:

Figura 9: Resumo estrutura do modelo redes neurais

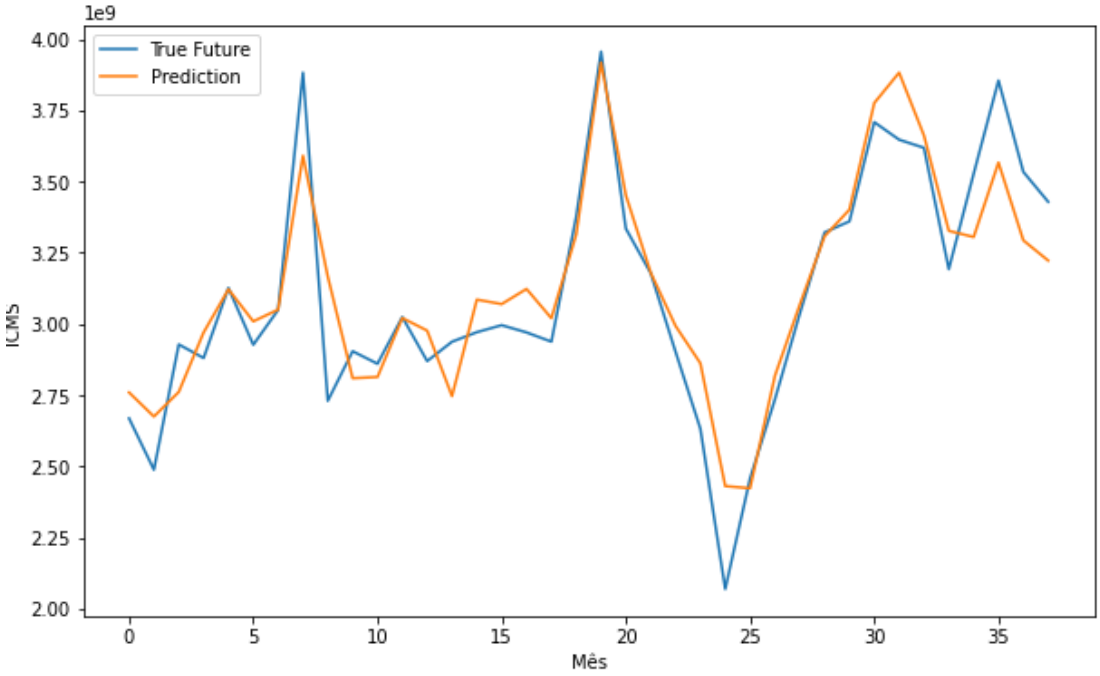
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 80)             29440
_________________________________________________________________
dropout (Dropout)            (None, 1, 80)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 80)                51520
_________________________________________________________________
dropout_1 (Dropout)          (None, 80)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 81
=================================================================
Total params: 81,041
Trainable params: 81,041
Non-trainable params: 0
_________________________________________________________________


LSTM:
Mean Absolute Error: 118070418.7881
Root Mean Square Error: 156068430.1167
```

Fonte: elaboração autores

A figura a seguir apresenta o comparativo entre os dados de teste, referentes aos anos de 2018, 2019 e 2020, e as previsões do modelo:

Figura 10: Comparativo entre a predição e a arrecadação efetiva.



Fonte: elaboração autores

A figura 10 apresenta que, a partir da base de treino, o modelo aprendeu com sucesso comportamento da série. Entretanto, é necessário destacar que parte da precisão deste teste – principalmente nas regiões de maior variação – deve-se ao fato de o modelo possuir lags e médias móveis em sua base de dados.

4.4. Análise dos Resultados

Posteriormente, confrontou-se os valores gerados e seus respectivos erros com a arrecadação efetivamente ocorrida no período, bem como outras metodologias de previsões já utilizadas pela Secretaria da Fazenda do Rio Grande do Sul. Neste ponto é importante fazer uma breve explicação sobre qual é a arrecadação efetivamente ocorrida que este artigo utiliza para análise. Ocasionalmente, a arrecadação efetiva do ICMS apresenta outliers que proporcionam grandes distorções na série temporal como, por

exemplo, a privatização da CEEE-D, gerando uma entrada excepcional de R$922 milhões de ICMS no mês de julho de 2021 (aproximadamente 26% da arrecadação total deste mês). Por óbvio, é inviável para um modelo baseado em informações do passado reagir a tais eventos. Ciente de tal condição, a proposta deste artigo visa verificar a precisão de uma linha base referente a arrecadação, um ponto de partida onde eventos discrepantes – que muitas vezes apresentam fácil previsibilidade por métodos empíricos – possam ser somados ou subtraídos da série calculada pelo modelo LSTM. A tabela 01 apresenta os dados supracitados trazendo resultados de outros métodos preditivos já utilizados pela SEFAZ/RS para análise comparativa. Ademais o mês de julho apresenta a correção referente a venda da CEEE-D, somada em ambas as previsões.

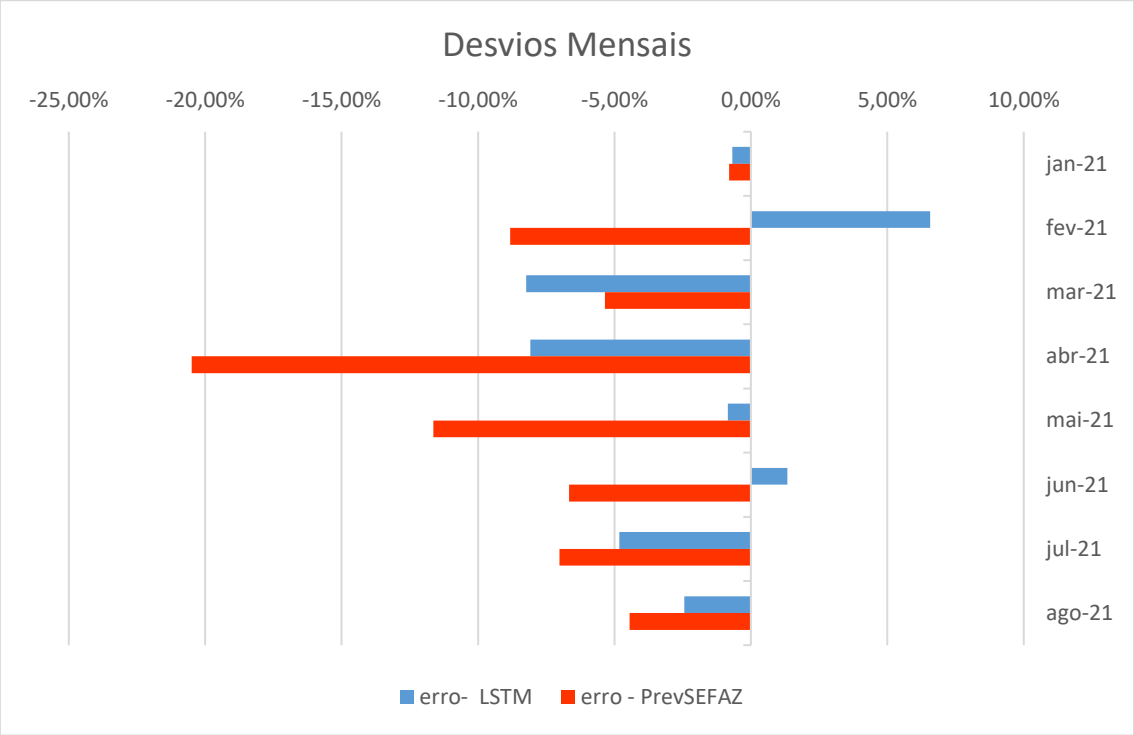
Tabela 02: Resultados obtidos dos modelos LSTM e Prev. SEFAZ.

Período	ICMS arrecadado	RNN - LSTM		PREVISÃO SEFAZ	
		ICMS - LSTM	erro- LSTM	ICMS - PrevSEFAZ	erro - PrevSEFAZ
jan-21	3.618.514,34	3.593.784,32	-0,68%	3.589.500,00	-0,80%
fev-21	3.192.257,37	3.401.825,28	6,56%	2.910.440,00	-8,83%
mar-21	3.527.156,46	3.236.530,18	-8,24%	3.338.340,00	-5,35%
abr-21	3.854.845,13	3.543.155,97	-8,09%	3.064.620,00	-20,50%
mai-21	3.534.051,89	3.504.327,17	-0,84%	3.122.650,00	-11,64%
jun-21	3.428.958,71	3.474.998,78	1,34%	3.200.390,00	-6,67%
jul-21	3.533.717,24	3.363.342,85	-4,82%	3.286.000,00	-7,01%
ago-21	3.791.069,57	3.698.511,10	-2,44%	3.622.340,00	-4,45%
total	28.480.570,70	27.816.475,65	-2,33%	26.134.280,00	-8,24%

Fonte: elaboração autores

Conforme os dados apresentados na tabela 01, é possível identificar 8 previsões mensais de janeiro de 2021 a agosto de 2021. O modelo utilizando redes neurais apresentou erro acumulado de -2,33%, frente ao erro de -8,24% apresentado por outros métodos preditivos já utilizados pela SEFAZ-RS. Ademais, a dispersão mensal máxima foi de -8,24% e -20,50% respectivamente.

Figura 11: Desvios mensais das previsões frente a arrecadação efetiva.



Fonte: elaboração autores

A figura 1 ilustra estes desvios mensais em relação à arrecadação efetivada no período. Nota-se que o modelo desenvolvido por redes neurais para este período manifesta, além de erros menores em módulo, uma menor tendência em seus valores, apresentando valores tanto positivos quanto negativos. Tal fato, por óbvio, contribui para a diminuição do erro total para o intervalo.

5. Conclusão

Este trabalho pode ser resumido em dois grandes objetivos. O primeiro é proporcionar novas metodologias para predição das receitas tributárias, mais precisamente a arrecadação do ICMS mensal do Estado do Rio Grande do Sul, demonstrando que é possível chegar em resultados consistentes através destas ferramentas. Neste aspecto, o modelo apresentou um erro acumulado de 2,33% em módulo para o período analisado, um resultado significativo frente a metodologias já consolidadas na SEFAZ-RS.

Já o segundo objetivo, quiçá de maior relevância, é garantir a transparência e possibilidade de aperfeiçoamento de novos métodos. Tal fato decorre que a utilização de redes neurais para este tipo de análise é ainda um campo incipiente, não existindo regulação técnica e normativa consolidada. Deste modo, com intuito de possibilitar a avaliação crítica e aperfeiçoamento por terceiros, é apresentado no anexo A deste artigo o código aberto utilizado para o desenvolvimento do Modelo em python. O campo de aprendizado de máquina é extremamente vasto, apresentando inúmeras bibliotecas distintas que podem oferecer soluções mais robustas para a base de dados utilizada. Esta tecnologia disruptiva permite soluções científicas ainda pouco exploradas no campo de previsão de arrecadação. Outras bibliotecas não exploradas neste artigo, a exemplo do Extreme Gradient Boosting (XGBoost), devem ser testadas e confrontadas com as previsões utilizadas atualmente. Outrossim, um ponto crucial de melhoria neste código é adaptar a arquitetura da rede criada para outros intervalos mais longos de predição.

Quanto as limitações do método proposto, devemos destacar sua complexidade e incertezas quanto a interiorização de mudanças de bruscas de patamares, a exemplo de alterações súbitas na série por fatores externos (mudança de alíquotas, recessão em função da pandemia etc.). A falta de regulamentação formal também acarreta desafios na definição de sua melhor arquitetura para cada problema. Ademais, compartilham as limitações intrínsecas de modelos uni variados.

Por fim, cabe destacar que não é objetivo deste trabalho abordar uma possível substituição dos modelos estatísticos consagrados por estes algoritmos que utilizam redes neurais. Na área de predição é de suma importância o trabalho conjunto de metodologias diferentes para resolver o mesmo problema, possibilitando uma análise crítica e objetiva dos resultados.

2. Referências

WEN Hao, CHEN Hao. (2020). *Tax Prediction Based on LSTM Recurrent Neural Network* School of Computer Science & Information Engineering,Hubei University,Wuhan 430062,China

http://www.jsjkx.com/EN/abstract/abstract19565.shtml

S. Siami-Namini, N. Tavakoli and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3285-3292, doi: 10.1109/BigData47090.2019.9005997.

Van Houdt, G., Mosquera, C. & Nápoles, G. A review on the long short-term memory model. Artif Intell Rev 53, 5929–5955 (2020). https://doi.org/10.1007/s10462-020-09838-1

Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

Reimers, Nils & Gurevych, Iryna. (2017). Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks.

ANEXO A – Código Python

ICMS-Artigo-AnexoA

November 26, 2021

1 Previsão de arrecadação do ICMS com a utilização de rede neural recorrente LSTM (Long Short-Term Memory) - previsão de julho 21

Fernando Schwartzer

Gabriel Zuanazzi Dornelles

Jacó Braatz

Importação das bibliotecas Pandas (biblioteca Python para análise de dados) e Numpy (biblioteca Python para realizar cálculos em Arranjos Multidimensionais)

[1]:
```
import pandas as pd
import numpy as np
```

Carregamento da planilha com os dados:

[2]:
```
df = pd.read_excel('icms-artigo-jun21.xlsx')
df
```

[2]:
```
          DATA          ICMS
0   2008-01-01  1.265624e+09
1   2008-02-01  1.138744e+09
2   2008-03-01  1.093966e+09
3   2008-04-01  1.200875e+09
4   2008-05-01  1.204678e+09
..         …           …
157 2021-02-01  3.192257e+09
158 2021-03-01  3.527156e+09
159 2021-04-01  3.854845e+09
160 2021-05-01  3.534052e+09
161 2021-06-01  3.428959e+09

[162 rows x 2 columns]
```

2 Feature Engineering

Remoção da primeira coluna, com as informações de ID:

[3]:
```
df['SMA(12)'] = df.ICMS.rolling(12).mean()
df['SMA(6)'] = df.ICMS.rolling(6).mean()
df['SMA(3)'] = df.ICMS.rolling(3).mean()
df['SMA(2)'] = df.ICMS.rolling(2).mean()
df
```

[3]:

	DATA	ICMS	SMA(12)	SMA(6)	SMA(3)	SMA(2)
0	2008-01-01	1.265624e+09	NaN	NaN	NaN	NaN
1	2008-02-01	1.138744e+09	NaN	NaN	NaN	1.202184e+09
2	2008-03-01	1.093966e+09	NaN	NaN	1.166111e+09	1.116355e+09
3	2008-04-01	1.200875e+09	NaN	NaN	1.144528e+09	1.147420e+09
4	2008-05-01	1.204678e+09	NaN	NaN	1.166506e+09	1.202776e+09
..	…	…	…	…	…	…
157	2021-02-01	3.192257e+09	3.056552e+09	3.474679e+09	3.485928e+09	3.405386e+09
158	2021-03-01	3.527156e+09	3.108646e+09	3.508974e+09	3.445976e+09	3.359707e+09
159	2021-04-01	3.854845e+09	3.210585e+09	3.591402e+09	3.524753e+09	3.691001e+09
160	2021-05-01	3.534052e+09	3.332685e+09	3.562306e+09	3.638684e+09	3.694449e+09
161	2021-06-01	3.428959e+09	3.413640e+09	3.525964e+09	3.605952e+09	3.481505e+09

[162 rows x 6 columns]

[4]:
```
df['lag(12)'] = df.ICMS.shift(12)
df['lag(6)'] = df.ICMS.shift(6)
df['lag(4)'] = df.ICMS.shift(4)
df['lag(3)'] = df.ICMS.shift(3)
df['lag(2)'] = df.ICMS.shift(2)
df['lag(1)'] = df.ICMS.shift(1)
df = df.dropna()
```

Visualização da planilha resultante:

[5]:
```
df
```

[5]:

	DATA	ICMS	SMA(12)	SMA(6)	SMA(3)	SMA(2)	lag(12)	lag(6)	lag(4)	lag(3)	lag(2)	lag(1)
12	2009-01-01	1.342779e+09	1.241859e+09	1.284620e+09	1.301309e+09	1.338301e+09	1.265624e+09	1.138744e+09	1.093966e+09	1.200875e+09	1.204678e+09	1.471824e+09

```
13  2009-02-01  1.111070e+09  1.239552e+09  1.274328e+09  1.218307e+09
14  2009-03-01  1.116238e+09  1.241409e+09  1.241900e+09  1.190029e+09
15  2009-04-01  1.282452e+09  1.248207e+09  1.235615e+09  1.169920e+09
16  2009-05-01  1.249666e+09  1.251956e+09  1.217213e+09  1.216119e+09
..         …           …           …           …           …
157 2021-02-01  3.192257e+09  3.056552e+09  3.474679e+09  3.485928e+09
158 2021-03-01  3.527156e+09  3.108646e+09  3.508974e+09  3.445976e+09
159 2021-04-01  3.854845e+09  3.210585e+09  3.591402e+09  3.524753e+09
160 2021-05-01  3.534052e+09  3.332685e+09  3.562306e+09  3.638684e+09
161 2021-06-01  3.428959e+09  3.413640e+09  3.525964e+09  3.605952e+09

           SMA(2)       lag(12)       lag(6)        lag(4)        lag(3)  \
12   1.271926e+09  1.265624e+09  1.199088e+09  1.310804e+09  1.320165e+09
13   1.226925e+09  1.138744e+09  1.172822e+09  1.320165e+09  1.360075e+09
14   1.113654e+09  1.093966e+09  1.310804e+09  1.360075e+09  1.201072e+09
15   1.199345e+09  1.200875e+09  1.320165e+09  1.201072e+09  1.342779e+09
16   1.266059e+09  1.204678e+09  1.360075e+09  1.342779e+09  1.111070e+09
..            …           …           …           …           …
157  3.405386e+09  3.178204e+09  3.035551e+09  3.360273e+09  3.708629e+09
158  3.359707e+09  2.902026e+09  3.321385e+09  3.708629e+09  3.647012e+09
159  3.691001e+09  2.631578e+09  3.360273e+09  3.647012e+09  3.618514e+09
160  3.694449e+09  2.068846e+09  3.708629e+09  3.618514e+09  3.192257e+09
161  3.481505e+09  2.457502e+09  3.647012e+09  3.192257e+09  3.527156e+09

           lag(2)        lag(1)
12   1.360075e+09  1.201072e+09
13   1.201072e+09  1.342779e+09
14   1.342779e+09  1.111070e+09
15   1.111070e+09  1.116238e+09
16   1.116238e+09  1.282452e+09
..            …           …
157  3.647012e+09  3.618514e+09
158  3.618514e+09  3.192257e+09
159  3.192257e+09  3.527156e+09
160  3.527156e+09  3.854845e+09
161  3.854845e+09  3.534052e+09

[150 rows x 12 columns]
```

Importação da biblioteca Datetime, que fornece as classes para manipulação de datas e horas:

[6]:
```
import datetime as dt
```

A função toordinal () retorna o ordinal gregoriano proléptico de uma data:

[ ]:
```
df['DATA'] = pd.to_datetime(df['DATA'])
df['DATA'] = df['DATA'].map(dt.datetime.toordinal)
```

Divisão da base de dados em uma base de treino, com os dados 75% iniciais, e uma base de testes

com os 25% restantes, mais recentes:

[8]:
```
train_size = int(len(df)*0.75)
train_dataset, test_dataset = df.iloc[:train_size], df.iloc[train_size:]
```

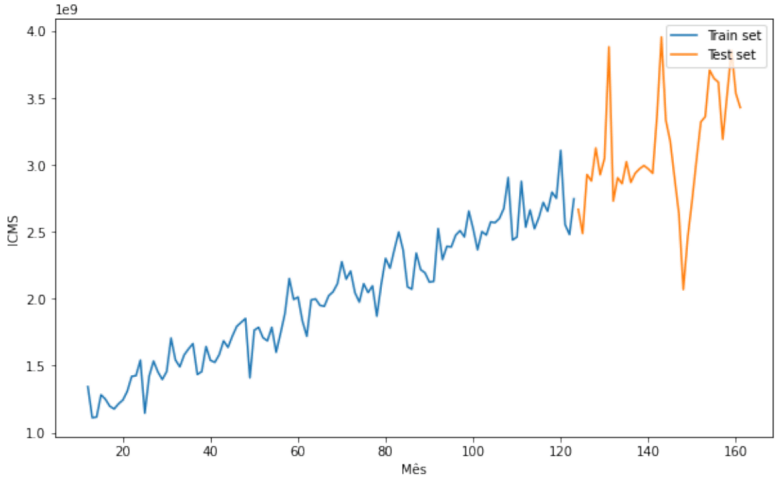
Importação da Matplotlib, biblioteca Python de plotagem 2d, que auxilia a biblioteca matemática NumPy:

[9]:
```
import matplotlib.pyplot as plt
```

Impressão das dimensões das bases de treino e test (Treino: 111 meses e 12 variáveis | Teste: 38 meses e 12 variáveis):

[10]:
```
plt.figure(figsize = (10, 6))
plt.plot(train_dataset.ICMS)
plt.plot(test_dataset.ICMS)
plt.xlabel('Mês')
plt.ylabel('ICMS')
plt.legend(['Train set', 'Test set'], loc='upper right')
print('Dimension of train data: ',train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

```
Dimension of train data:  (112, 12)
Dimension of test data:  (38, 12)
```



Separação da variável dependente das independentes, tanto na base de treino, quanto na de testes:

[11]:
```
X_train = train_dataset.drop('ICMS', axis = 1)
y_train = train_dataset.loc[:,['ICMS']]
```

[12]:
```
X_test = test_dataset.drop('ICMS', axis = 1)
y_test = test_dataset.loc[:,['ICMS']]
```

[13]:
```
print('X_train.shape: ', X_test.shape)
print('y_train.shape: ', y_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_test.shape: ', y_train.shape)
```

```
X_train.shape:  (38, 11)
y_train.shape:  (112, 1)
X_test.shape:  (38, 11)
y_test.shape:  (112, 1)
```

Importação da ferramenta de pré-processamento de dados MinMaxScaler da biblioteca scikit-learn (originalmente scikits.learn), que é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.Inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, gradient boosting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy. O MinMaxScaler dimensiona e traduz cada característica individualmente de modo que esteja na faixa dada no conjunto de treinamento, por exemplo, entre zero e um.

[14]:
```
from sklearn.preprocessing import MinMaxScaler
```

[15]:
```
scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))
```

[16]:
```
input_scaler = scaler_x.fit(X_train)
output_scaler = scaler_y.fit(y_train)
```

[17]:
```
train_y_norm = output_scaler.transform(y_train)
train_x_norm = input_scaler.transform(X_train)
```

[18]:
```
test_y_norm = output_scaler.transform(y_test)
test_x_norm = input_scaler.transform(X_test)
```

[19]:
```
print('train_y_norm.shape: ', train_y_norm.shape)
print('train_x_norm.shape: ', train_x_norm.shape)
print('test_y_norm.shape: ', test_y_norm.shape)
print('test_x_norm.shape: ', test_x_norm.shape)
```

```
train_y_norm.shape:  (112, 1)
train_x_norm.shape:  (112, 11)
test_y_norm.shape:  (38, 1)
test_x_norm.shape:  (38, 11)
```

Redimensionamento da matriz para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A primeira dimensão representa o tamanho do

lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

[20]:
```
X_test = test_x_norm.reshape(38, 1, 11)
X_train = train_x_norm.reshape(112, 1, 11)
y_test = test_y_norm.reshape(38,1)
y_train = train_y_norm.reshape(112, 1)
```

Importação da TensorFlow, plataforma completa de código aberto para machine learning, além do numpy.mean, para computar média aritmética, numpy.std, para computar o desvio padrão:

[21]:
```
import tensorflow as tf
from numpy import mean
from numpy import std
from matplotlib import pyplot
```

Ajuste de modelo com 32, 64 e 128 neurônios para verificação do mais adequado para a aplicação. É buscado o parâmetro que retorne o menor erro, com a menor variação.

[ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, neurons):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = neurons, return_sequences = True,␣
 ↪input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.LSTM(units = neurons))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 100, validation_split = 0.2,␣
 ↪batch_size = 4, shuffle = False)
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size=4, verbose=0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')
```

```
# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [32, 64, 128]
run_experiment(n_params)
```

[ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, neurons):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = neurons, return_sequences = True,␣
 ↪input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.LSTM(units = neurons))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 50, validation_split = 0.2, batch_size␣
 ↪= 4, shuffle = False)
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size=4, verbose=0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')
```

```
# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [60, 65, 70, 75, 80, 85, 90]
run_experiment(n_params)
```

Ajuste de modelo com Tamanho do Lote 1, 2, 3, 4, 5, 6, 7 e 8 para verificação do mais adequado para a aplicação. Tamanho do lote (Batch Size) é um termo usado em aprendizado de máquina e refere-se ao número de exemplos de treinamento usados em uma iteração

[ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, batch_size):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 80, return_sequences = True,␣
 ↪input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.LSTM(units = 80))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 50, validation_split = 0.2, batch_size␣
 ↪= batch_size, shuffle = False)
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
```

```
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')

# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [1, 2, 3, 4, 5, 6, 7, 8]
run_experiment(n_params)
```

Ajuste de modelo com diluição (Dropout) 0.05, 0.1, 0.2 e 0.3 para verificação do mais adequado para a aplicação. A diluição é uma técnica de regularização para reduzir o sobreajuste em redes neurais artificiais, impedindo co-adaptações complexas nos dados de treinamento.

[ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, dropout):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 80, return_sequences = True,␣
 ↪input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.LSTM(units = 80))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 50, validation_split = 0.2, batch_size␣
 ↪= 5, shuffle = False)
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size = 5, verbose = 0)
    return loss
```

```
# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')

# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [0.05, 0.1, 0.2, 0.3]
run_experiment(n_params)
```

Ajuste de modelo com os parâmetros verificados mais adequados nos testes anteriores:

[ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 80, return_sequences = True,␣
 ↪input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.LSTM(units = 80))
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 150, validation_split = 0.2,␣
 ↪batch_size = 5, shuffle = False)
    # Validando o modelo
```

```
    loss = model.evaluate(X_test, y_test, batch_size = 5, verbose = 0)
    return loss

# Resumindo as pontuações
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Loss: %.3f%% (+/-%.3f)' % (m, s))

# Rodando um experimento
def run_experiment(repeats=10):
    # Repetindo o experimento
    scores = list()
    for r in range(repeats):
        score = evaluate_model(X_train, y_train, X_test, y_test)
        score = score
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # Resumindo os resultados
    summarize_results(scores)

# Rodando o experimento
run_experiment()
```

[23]:
```
def create_model(units, dropout):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = units, return_sequences = True,
                input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.LSTM(units = units))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    return model
```

[24]:
```
model_lstm = create_model(80, 0.05)
```

[25]:
```
def fit_model(model):
    early_stop = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss',␣
 ↪patience = 10)
    history = model.fit(X_train, y_train, epochs = 150, validation_split = 0.2,␣
 ↪batch_size = 5, shuffle = False, callbacks = [early_stop])
    return history
```

[ ]:
```
history_lstm = fit_model(model_lstm)
```

Salvamento do modelo para posterior uso:

[ ]:
```
model_lstm.save('ICMS-artigo-jun')
```

Carregamento do modelo salvo:

[28]:
```
model_lstm = tf.keras.models.load_model('ICMS-artigo-jun')
```

Pré-processamento e predição da base de teste com a utilização do modelo carregado:

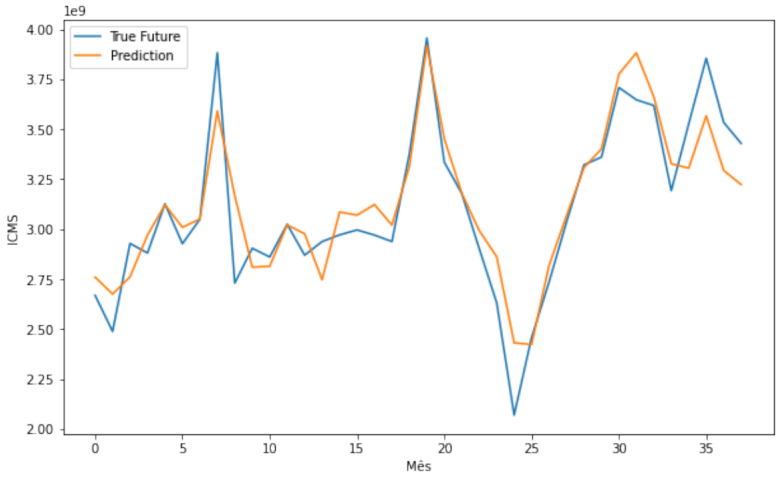
[29]:
```
y_test = scaler_y.inverse_transform(y_test)
y_train = scaler_y.inverse_transform(y_train)
```

[30]:
```
def prediction(model):
    prediction = model.predict(X_test)
    prediction = scaler_y.inverse_transform(prediction)
    return prediction
```

[31]:
```
prediction_lstm = prediction(model_lstm)
```

[32]:
```
def plot_future(prediction, y_test):
    plt.figure(figsize=(10, 6))
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(y_test), label='True Future')
    plt.plot(np.arange(range_future),np.array(prediction), label='Prediction')
    plt.legend(loc='upper left')
    plt.xlabel('Mês')
    plt.ylabel('ICMS')
```

[33]:
```
plot_future(prediction_lstm, y_test)
```



Cálculo do Erro Médio Absoluto e Raiz Quadrática Média:

[34]:
```
def evaluate_prediction(predictions, actual, model_name):
    errors = predictions - actual
    mse = np.square(errors).mean()
    rmse = np.sqrt(mse)
    mae = np.abs(errors).mean()
    print(model_name + ':')
    print('Mean Absolute Error: {:.4f}'.format(mae))
    print('Root Mean Square Error: {:.4f}'.format(rmse))
    print('')
```

[35]:
```
evaluate_prediction(prediction_lstm, y_test, 'LSTM')
```

```
LSTM:
Mean Absolute Error: 125268029.4021
Root Mean Square Error: 162546979.6234
```

Pré-processamento dos dados e aplicação do modelo em toda a base de dados (treino + teste):

[36]:
```
X = df.drop('ICMS', axis = 1)
y = df.loc[:,['ICMS']]
```

[37]:
```
y_norm = output_scaler.transform(y)
x_norm = input_scaler.transform(X)
```

[38]:
```
X = x_norm.reshape(150, 1, 11)
y = y_norm.reshape(150,1)
```

[39]:
```
y = scaler_y.inverse_transform(y)
```

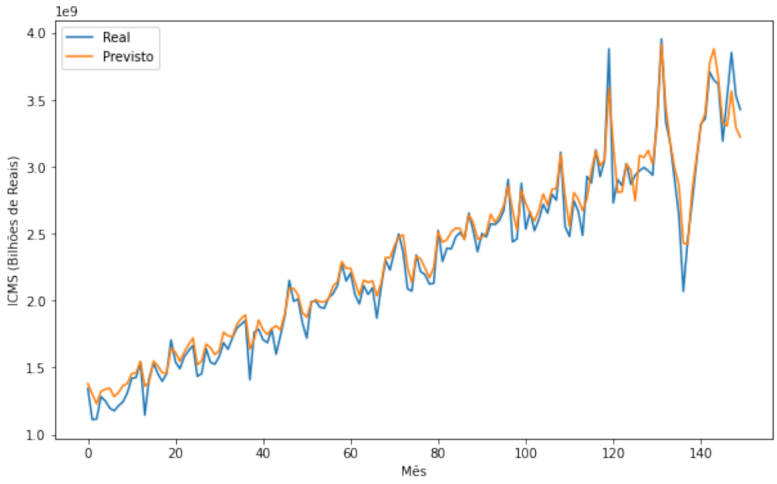
[40]:
```
def prediction(model):
    prediction = model.predict(X)
    prediction = scaler_y.inverse_transform(prediction)
    return prediction
```

[41]:
```
prediction_lstm = prediction(model_lstm)
```

[42]:
```
def plot_future(prediction, y):
    plt.figure(figsize=(10, 6))
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(y), label='Real')
    plt.plot(np.arange(range_future),np.array(prediction), label='Previsto')
    plt.legend(loc='upper left')
    plt.xlabel('Mês')
```

```
    plt.ylabel('ICMS (Bilhões de Reais)')
```

[43]:
```
plot_future(prediction_lstm, y)
```



Cálculo do erro médio percentual:

[44]:
```
real = y.flatten()
previsto = prediction_lstm.flatten()
```

[45]:
```
tabela = pd.DataFrame([real, previsto]).T
```

[46]:
```
tabela = tabela.rename(columns={0: "Real", 1: "Previsto"})
```

[47]:
```
tabela['Diferença'] = 1 - (tabela['Real'] / tabela['Previsto'])
```

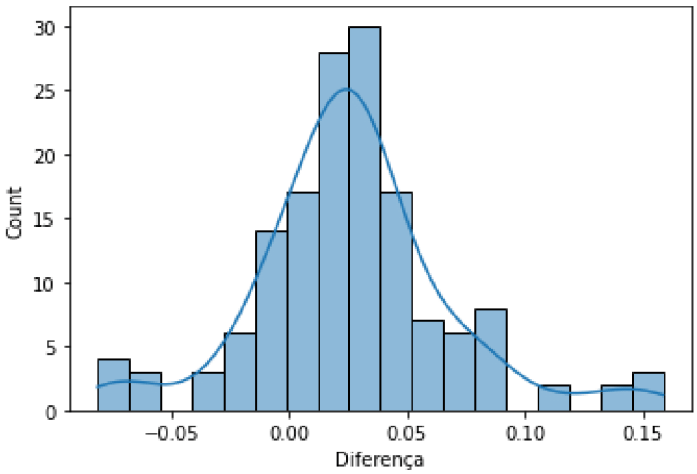
[48]:
```
print(tabela['Diferença'].mean())
```

```
0.026293330165477347
```

Distribuição dos erros:

[51]:
```
import seaborn as sns
sns.histplot(data=tabela, x="Diferença", kde=True)
```

[51]:
```
<AxesSubplot:xlabel='Diferença', ylabel='Count'>
```



[52]:
```
previsao = df
```

[53]:
```
periodo = 1 #escolher quantos passos a frente a previsão executa
for i in range(periodo):
    row = pd.DataFrame([], columns = previsao.columns)
    row.loc[0, 'SMA(12)'] = previsao.ICMS.iloc[-12:].mean()
    row.loc[0, 'SMA(6)'] = previsao.ICMS.iloc[-6:].mean()
    row.loc[0, 'SMA(3)'] = previsao.ICMS.iloc[-3:].mean()
    row.loc[0, 'SMA(2)'] = previsao.ICMS.iloc[-2:].mean()
    row.loc[0, 'lag(12)'] = previsao.ICMS.iloc[-12]
    row.loc[0, 'lag(6)'] = previsao.ICMS.iloc[-6]
    row.loc[0, 'lag(4)'] = previsao.ICMS.iloc[-4]
    row.loc[0, 'lag(3)'] = previsao.ICMS.iloc[-3]
    row.loc[0, 'lag(2)'] = previsao.ICMS.iloc[-2]
    row.loc[0, 'lag(1)'] = previsao.ICMS.iloc[-1]
    row.loc[0, 'DATA'] = previsao.DATA.iloc[-1]+1
    row = row.drop(['ICMS'], axis = 1)
    row = np.array(row.iloc[-1]).reshape(1, -1)
    row_norm = input_scaler.transform(row)
    to_prev = row_norm.reshape(1, 1, 11)
    prev = model_lstm.predict(to_prev)
    prev = scaler_y.inverse_transform(prev)
    row_ = pd.DataFrame(row, columns = ['DATA', 'SMA(12)', 'SMA(6)', 'SMA(3)',␣
↪'SMA(2)', 'lag(12)','lag(6)', 'lag(4)', 'lag(3)', 'lag(2)', 'lag(1)'])
```

```
    row_.loc[0, 'ICMS'] = prev[0]
    previsao = previsao.append(row_)
```

[ ]: