PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Rafael Santos Carvalho

Predição da arrecadação de tributos: um comparativo entre o *Facebook*™ *Prophet* e as redes neurais *Long Short-Term Memory*

Varginha, Minas Gerais

2021

Rafael Santos Carvalho

PREDIÇÃO DA ARRECADAÇÃO DE TRIBUTOS: UM COMPARATIVO ENTRE O *FACEBOOK™ PROPHET* E AS REDES NEURAIS *LONG SHORT-TERM MEMORY*

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.

Varginha, Minas Gerais

2021

LISTA DE ABREVIATURAS E SIGLAS

Bacen – Banco Central do Brasil
CAGED – Cadastro Geral de Empregados e Desempregados
EMA – Erro Médio Absoluto
ICMS – Imposto sobre Circulação de Mercadorias e Serviços
IGP-M – Índice Geral de Preços - Mercado
IPVA – Imposto sobre a Propriedade de Veículos Automotores
ITCD – Imposto sobre Transmissão Causa mortis e Doação
LDO – Lei de Diretrizes Orçamentárias
LOA – Lei Orçamentária Anual
LRF – Lei de Responsabilidade Fiscal
LSTM – *Long Short-Term Memory*
MAE – *Mean Absolut Error*
MLE – Maximum-likelihood Estimation
MLP – *Multilayer Perceptron*
OHE – *One Hot Encoder*
PIB – Produto Interno Bruto
RS – Rio Grande do Sul
SPC – Serviço de Proteção ao Crédito

SUMÁRIO

1. Introdução

1.1. Contextualização

Todo ente, seja ele município, estado federado ou a própria União possui o dever de realizar suas despesas de maneira harmônica com suas receitas, dever que, caso não cumprido, pode levar o ente a problemas fiscais como inadimplência de suas dívidas com fornecedores, atrasos dos pagamentos de aposentadorias, pensões e até mesmo dos salários do próprio funcionalismo. Além da própria crise fiscal que se origina da execução de despesas em montante superior ao das receitas, pode ainda o gestor público responder por crime de responsabilidade, conforme preceitua os artigos 10 e 11 da Lei 1.079/1950:

> "CAPÍTULO VI
> DOS CRIMES CONTRA A LEI ORÇAMENTÁRIA
>
> Art. 10. São crimes de responsabilidade contra a lei orçamentária:
> [...]
> 2 - Exceder ou transportar, sem autorização legal, as verbas do orçamento;
> [...]
> 10 - captar recursos a título de antecipação de receita de tributo ou contribuição cujo fato gerador ainda não tenha ocorrido;
> [...]
>
> CAPÍTULO VII
> DOS CRIMES CONTRA A GUARDA E LEGAL EMPREGO DOS DINHEIROS PÚBLICOS:
>
> Art. 11. São crimes contra a guarda e legal emprego dos dinheiros públicos:
> 1 - ordenar despesas não autorizadas por lei ou sem observância das prescrições legais relativas às mesmas; [...]" (Brasil, 1950).

Já as Leis de Diretrizes Orçamentárias (LDOs) surgem para orientar a elaboração das Leis Orçamentárias Anuais (LOAs), estas que, por sua vez, determinam o orçamento do ente para determinado ano. Conforme a Lei de Responsabilidade Fiscal (LRF) cabem às LDOs:

> "Art. 4º A lei de diretrizes orçamentárias atenderá o disposto no § 2o do art. 165 da Constituição e:
> I - disporá também sobre:
> a) equilíbrio entre receitas e despesas;
> b) critérios e forma de limitação de empenho, a ser efetivada nas hipóteses previstas na alínea b do inciso II deste artigo, no art. 9o e no inciso II do § 1o do art. 31;
> c) (VETADO)
> d) (VETADO)
> e) normas relativas ao controle de custos e à avaliação dos resultados dos programas financiados com recursos dos orçamentos;

> f) demais condições e exigências para transferências de recursos a entidades públicas e privadas;
> II - (VETADO)
> III - (VETADO)
> § 1o Integrará o projeto de lei de diretrizes orçamentárias Anexo de Metas Fiscais, em que serão estabelecidas metas anuais, em valores correntes e constantes, relativas a receitas, despesas, resultados nominal e primário e montante da dívida pública, para o exercício a que se referirem e para os dois seguintes.
> § 2o O Anexo conterá, ainda:
> I - avaliação do cumprimento das metas relativas ao ano anterior;
> II - demonstrativo das metas anuais, instruído com memória e metodologia de cálculo que justifiquem os resultados pretendidos, comparando-as com as fixadas nos três exercícios anteriores, e evidenciando a consistência delas com as premissas e os objetivos da política econômica nacional;
> III - evolução do patrimônio líquido, também nos últimos três exercícios, destacando a origem e a aplicação dos recursos obtidos com a alienação de ativos;
> IV - avaliação da situação financeira e atuarial:
> a) dos regimes geral de previdência social e próprio dos servidores públicos e do Fundo de Amparo ao Trabalhador;
> b) dos demais fundos públicos e programas estatais de natureza atuarial;
> V - demonstrativo da estimativa e compensação da renúncia de receita e da margem de expansão das despesas obrigatórias de caráter continuado.
> § 3o A lei de diretrizes orçamentárias conterá Anexo de Riscos Fiscais, onde serão avaliados os passivos contingentes e outros riscos capazes de afetar as contas públicas, informando as providências a serem tomadas, caso se concretizem.
> § 4o A mensagem que encaminhar o projeto da União apresentará, em anexo específico, os objetivos das políticas monetária, creditícia e cambial, bem como os parâmetros e as projeções para seus principais agregados e variáveis, e ainda as metas de inflação, para o exercício subsequente.” (Brasil, Lei Complementar 101/2000 – Lei de Responsabilidade Fiscal)

Percebe-se, portanto, que os gestores públicos estão sujeitos a diversos controles legais que visam a evitar que as despesas superem as receitas, estabelecendo metas, bem como a avaliação do cumprimento das metas estabelecidas em exercícios anteriores.

Sendo assim, a projeção das receitas é de suma importância para elaboração das Leis de Diretrizes Orçamentárias e das Leis Orçamentárias Anuais, bem como para a avaliação contínua pela Administração Pública da execução das despesas, com vistas a equilibrá-la à projeção de receitas, evitando *déficits* fiscais e infrações à Lei de Responsabilidade Fiscal.

1.2. O problema proposto

Para avaliar continuamente se as despesas programadas poderão efetivamente ser executadas sem gerar *déficit* fiscal, os gestores públicos necessitam de ferramentas que permitam predizer a arrecadação futura para tomarem decisões acerca da continuidade, suspensão ou até mesmo o cancelamento da execução de

despesas. Geralmente as administrações tributárias realizam estimativas trimestrais e, quando necessitam de uma previsão em uma janela menor de tempo, utilizam previsões mensais. Muitas destas estimativas têm como objetivo ajustar as despesas a curto/médio prazo, bem como dar andamento a medidas burocráticas de aumento de receita, como solicitação de crédito suplementar, emissão de títulos da dívida pública e aumento de tributos, que necessitam de maior tempo para surtirem efeito, uma vez que estão quase sempre sujeitas ao processo legislativo.

Este trabalho visa encontrar métodos de predição em períodos mais curtos que os tradicionais, permitindo que gestores públicos prevejam a arrecadação em curtíssimo prazo (diariamente), permitindo tomada de decisões em caráter emergencial, principalmente no tocante à execução das despesas discricionárias, visando ao equilíbrio das contas públicas dos entes federados.

Para isso, serão utilizados dois algoritmos de predição de séries temporais (Facebook™ Prophet e redes neurais Long Short-Term Memory – LSTM) para analisar os dados diários de arrecadação tributária de três tributos estaduais do Rio Grande do Sul (Imposto sobre a Propriedade de Veículos Automotores – IPVA, Imposto sobre Transmissão Causa mortis e Doação – ITCD e Imposto sobre Circulação de Mercadorias e Serviços – ICMS) e estimar a arrecadação em datas futuras. Segundo dados do Portal da Transparência[1] do Estado do Rio Grande do Sul, em 2020 estes três tributos foram responsáveis por 63,53% da arrecadação estadual.

Receita	**Valor**		**Percentual**
ICMS	R$	36.380.727.217,60	57,27%
IPVA	R$	3.219.137.802,00	5,07%
ITCD	R$	759.805.762,52	1,20%
Contribuição Previdenciária	R$	18.933.532.080,80	29,80%
Outras Receitas	R$	4.231.732.386,74	6,66%
Total	**R$**	**63.524.935.249,66**	**100,00%**

Figura 1 – Valores e proporções da arrecadação por fonte de receita

Em um segundo momento, dados econômicos do estado do Rio Grande do Sul e nacionais serão agregados ao *dataset* dos dados de arrecadação de ICMS, permitindo avaliar a eficácia destas novas variáveis preditoras para a redução do erro dos modelos.

1 Disponível em: <http://www.transparencia.rs.gov.br/QvAJAXZfc/opendoc.htm?document=Transparencia.qvw &host= QVS%40QLVPRO06&anonymous=true>. Acesso em 7 de abril de 2021.

2. Coleta de dados

Os dados referentes à arrecadação diária de tributos do estado do Rio Grande do Sul foram coletados do Receita Dados[2], portal de dados da Secretaria Estadual de Fazenda do estado do Rio Grande do Sul. Já os dados do IGP-M para atualização dos valores dos demais *datasets* a valor presente (quando necessário) foram coletados do *IpeaData*[3].

Quando da agregação de novos dados ao *dataset* de arrecadação, os dados serão também obtidos do *IpeaData*, sendo os dados do PIB trimestral do Rio Grande do Sul e de admissões e demissões mantidos na mesma escala que os coletados, uma vez que os dados de emprego, por não se tratarem de valores monetários, não estão sujeitos a alterações com o passar do tempo e os dados do PIB trimestral do estado já estão expressos em valores reais.

Os métodos que realizam as coletas de dados estão na classe *DownloadDados*, dentro de arquivo de mesmo nome, e abrangem leitura de arquivos CSV (arrecadação mensal, PIB mensal do Brasil e dados de admissões e demissões), leitura de arquivos do Microsoft Excel (PIB trimestral do RS) e *webscrapping* (IGP-M mensal).

3. Tratamento de dados

Uma vez que os valores coletados são nominais, estes serão ajustados a valor presente com base no IGP-M, índice que melhor reflete as alterações gerais de preço de mercado, tanto para consumidores (no caso do governo quando adquire produtos e serviços), quanto para produtores (no caso dos contribuintes, que pagam os tributos – principalmente o ICMS – sobre bases de cálculo que geralmente estão correlacionadas ao valor de mercado do bem, produto ou serviço). Sendo assim, a atualização dos valores de arrecadação pelo IGP-M permite a verificação da arrecadação real tanto pela ótica dos contribuintes quanto pela ótica do governo. Caso os valores não fossem atualizados a valor presente, os modelos de aprendizado de máquina teriam que lidar, além das variações reais da arrecadação, com as variações da inflação, o que provavelmente aumentaria o erro médio das predições.

---

[2] Disponível em: <http://receitadados.fazenda.rs.gov.br/Arquivos/Arrecadação Diária.csv >. Acesso em 8 de abril de 2021.

[3] Disponível em: http://www.ipeadata.gov.br/ExibeSerie.aspx?stub=1&serid37796=37796&serid36482=36482. Acesso em 8 de abril de 2021.

Já os valores do PIB nacional mensal foram ajustados pelo IGP-M da mesma forma que os dados de arrecadação, uma vez que da maneira que foram coletados estavam expressos em termos nominais.

Na coleta de dados do PIB nacional, foram necessários pequenos ajustes na exibição das datas (ajustes realizados nos métodos que fazem o *download* dos dados), que traziam dados do mês de outubro como sendo de janeiro. Já na coleta de dados de emprego do CAGED (admissões e demissões), além do ajuste das datas foi necessário o preenchimento de alguns campos que retornaram *NaN* (base de dados antiga), este preenchimento foi realizado com base no *download* de um terceiro *dataset* que continha os dados do saldo mensal de contratações, permitindo deduzir o valor de admissões ou demissões que estivessem sem valores válidos com base na subtração do saldo pelo campo (admissão ou demissão) que contivesse valores válidos para determinado mês.

```
@staticmethod
def download_dados_emprego():
  """ Baixa os dados de emprego do IPEA Data """

  # Baixa os dados de saldo de empregos para preencher campos que vierem
sem valores válidos
  url_saldo_caged_antigo =
'http://www.ipeadata.gov.br/ExibeSerie.aspx?oper=exportCSVBr&serid272844966
=272844966&serid272844966=272844966'
  df_saldo_caged_antigo = pd.read_csv(url_saldo_caged_antigo, sep=';')
  df_saldo_caged_antigo['Data'] =
df_saldo_caged_antigo['Data'].astype(str).str.replace('.',
'/').replace('0000', '').str.replace('/1$', '/10')
  df_saldo_caged_antigo =
df_saldo_caged_antigo.drop(df_saldo_caged_antigo.columns[2], axis=1)
  df_saldo_caged_antigo.columns = ['Data', 'Saldo']

  # Dados antigos do CAGED (Admissões e Demissões)
  url_caged_antigo =
'http://www.ipeadata.gov.br/ExibeSerie.aspx?oper=exportCSVBr&serid231410417
=231410417&serid231410418=231410418'
  df_antigo = pd.read_csv(url_caged_antigo, sep=';')
  df_antigo['Data'] = df_antigo['Data'].astype(str).str.replace('.',
'/').replace('0000', '').str.replace('/1$', '/10')
  df_antigo = df_antigo.drop(df_antigo.columns[3], axis=1)
  df_antigo.columns = ['Data', 'Admissoes', 'Demissoes']
  datas_nan = df_antigo['Data'][np.isnan(df_antigo['Demissoes'])==True] #
Datas cujo valor de demissões é NaN
  saldos_nan =
df_saldo_caged_antigo[df_saldo_caged_antigo['Data'].isin(pd.DataFrame(datas
_nan)['Data'])] # Saldos em datas cujo valor de demissões é NaN
  df_antigo['Demissoes'][np.isnan(df_antigo['Demissoes'])==True] =
df_antigo['Admissoes'][np.isnan(df_antigo['Demissoes'])==True]-
saldos_nan['Saldo']
  df_antigo['Demissoes'] = df_antigo['Demissoes'].astype(int)

  # Dados novos do CAGED (Admissões e Demissões)
  url_caged_novo =
'http://www.ipeadata.gov.br/ExibeSerie.aspx?oper=exportCSVBr&serid209672533
```

```
4=2096725334&serid2096725335=2096725335'
  df_novo = pd.read_csv(url_caged_novo, sep=';')
  df_novo['Data'] = df_novo['Data'].astype(str).str.replace('.',
'/').replace('0000', '').str.replace('/1$', '/10')
  df_novo = df_novo.drop(df_novo.columns[3], axis=1)
  df_novo.columns = ['Data', 'Admissoes', 'Demissoes']

  df_final = df_novo.iloc[::-1].append(df_antigo.iloc[::-
1]).reset_index(drop=True)

  for i in range(0, len(df_final)):
    ano = df_final['Data'].str.split('/')[i][0]
    mes = df_final['Data'].str.split('/')[i][1]
    df_final['Data'].loc[i] = mes+'/'+ano

  return df_final
```

Figura 2 – Download e tratamento de dados do CAGED

Após o tratamento dos dados antigos e novos foi realizada a junção dos dois *dataframes*, retornando um único Pandas *dataframe* como resultado.

Já as atualizações dos valores a valor presente são realizadas pelos métodos *corrige_inflacao* e *corrige_inflacao_pib* da classe *CorrigeValores* do arquivo *Util.py*, que verificam o período a que se referem os dados do PIB e da arrecadação e multiplicam pelo número índice, trazendo os valores a valor presente da última data disponível da série histórica do IGP-M.

```
@staticmethod
def corrige_inflacao(pd_tributo, pd_inflacao):
  """ Dado um dataframe com os números-índice para correção e outro
dataframe com os valores de tributo,
  corrige estes trazendo-os ao valor presente da última data disponível no
dataframe de valores de tributo."""

  ultimo_mes_inflacao = pd_inflacao.loc[0, 'Mes/Ano']
  ultimo_numero_indice = pd_inflacao.loc[0, 'Numero Indice']

  for indice, linha in pd_tributo.iterrows():
    mes_atual = linha['Data'].strftime("%m/%Y")

    ''' Nos meses anteriores à última data publicada para a inflação, o
valor do tributo será
    multiplicado pela razão entre o último número-índice e o número índice
correspondente ao mês seguinte ao do
    tributo. '''
    if datetime.strptime(mes_atual, '%m/%Y').date() <
datetime.strptime(ultimo_mes_inflacao, '%m/%Y').date():
      numero_indice_atual = pd_inflacao[pd_inflacao['Mes/Ano'] ==
mes_atual]['Numero Indice'].item()
    elif datetime.strptime(mes_atual, '%m/%Y').date() >=
datetime.strptime(ultimo_mes_inflacao, '%m/%Y').date():
      numero_indice_atual = ultimo_numero_indice

    pd_tributo.loc[indice, 'Valor'] = pd_tributo.loc[indice, 'Valor'] *
ultimo_numero_indice / numero_indice_atual

  return pd_tributo
```

```
@staticmethod
def corrige_inflacao_pib(pd_pib, pd_inflacao):
  """ Dado um dataframe com os números-índice para correção e outro
dataframe com os valores do PIB Nominal,
  corrige estes trazendo-os ao valor presente da última data disponível no
dataframe de valores de PIB."""

  ultimo_mes_inflacao = pd_inflacao.loc[0, 'Mes/Ano']
  ultimo_numero_indice = pd_inflacao.loc[0, 'Numero Indice']

  for indice, linha in pd_pib.iterrows():
    mes_atual = linha['Data']

    ''' Nos meses anteriores à última data publicada para a inflação, o
valor do tributo será
    multiplicado pela razão entre o último número-índice e o número índice
correspondente ao mês seguinte ao do
    tributo. '''
    if datetime.strptime(mes_atual, '%m/%Y').date() <
datetime.strptime(ultimo_mes_inflacao, '%m/%Y').date():
      numero_indice_atual = pd_inflacao[pd_inflacao['Mes/Ano'] ==
mes_atual]['Numero Indice'].item()
    elif datetime.strptime(mes_atual, '%m/%Y').date() >=
datetime.strptime(ultimo_mes_inflacao, '%m/%Y').date():
      numero_indice_atual = ultimo_numero_indice

    pd_pib.loc[indice, 'PIB'] = pd_pib.loc[indice, 'PIB'] *
ultimo_numero_indice / numero_indice_atual

  return pd_pib
```

Figura 3 – Métodos de atualização de valores a valor presente

4. Análise e Exploração dos Dados

4.2. Os dados nominais

No primeiro momento serão analisados os dados nominais dos tributos estaduais, com o objetivo de comparar com os dados reais, estes que serão efetivamente utilizados para treinamento e predição nos modelos. Para tanto, seguem abaixo os dados descritivos das séries temporais da arrecadação nominal dos três tributos estaduais:

Medida	ICMS	IPVA	ITCD
Quantidade	3.167,00	3.119,00	3.097,00
Média	104.326.511,37	9.263.437,50	1.709.255,15
Desvio Padrão (σ)	145.973.777,35	18.156.530,23	2.582.832,55
Mínimo	-153.507,11	-167,22	-1.375,75
25%	16.454.912,34	1.829.823,52	612.259,54
50%	39.408.950,31	4.662.392,00	1.184.296,94

75%	145.005.742,05	9.745.511,49	2.011.134,15
Máximo	1.164.392.552,98	294.336.676,91	62.862.554,05

Figura 4 – Descrição da arrecadação nominal dos tributos estaduais

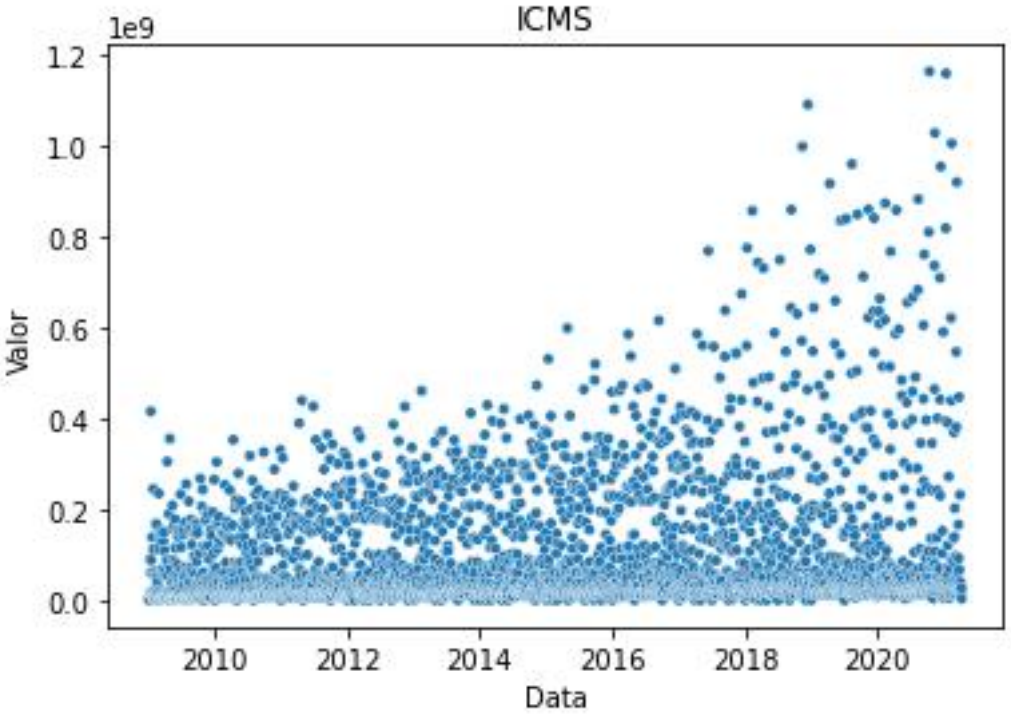

Figura 5 – Arrecadação nominal do ICMS do RS

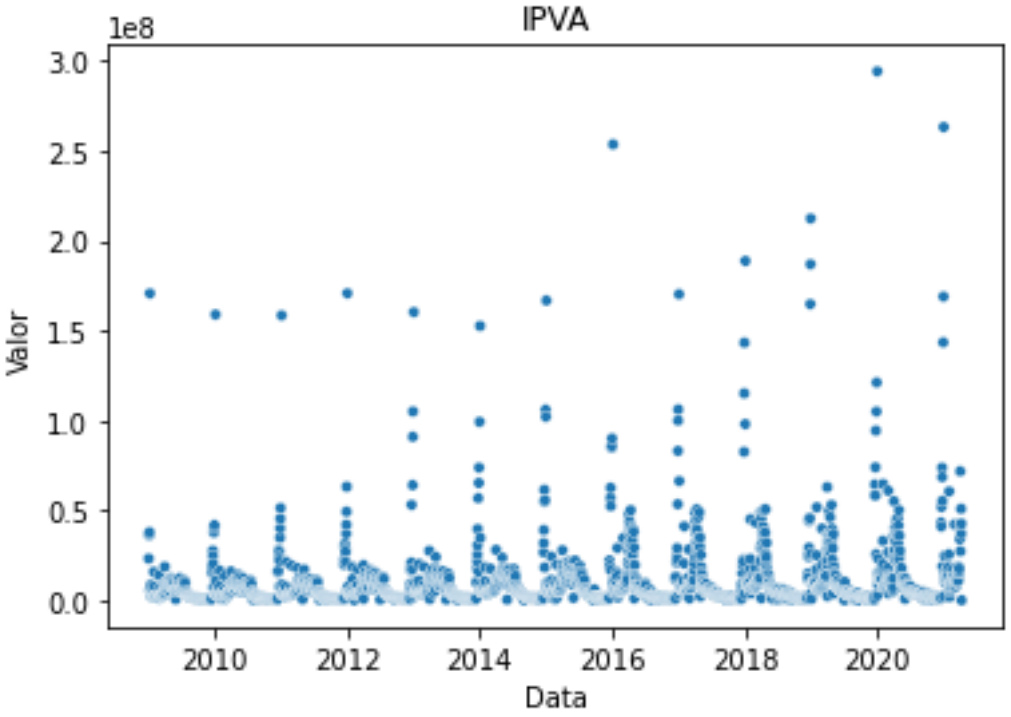

Figura 6 – Arrecadação nominal do IPVA do RS

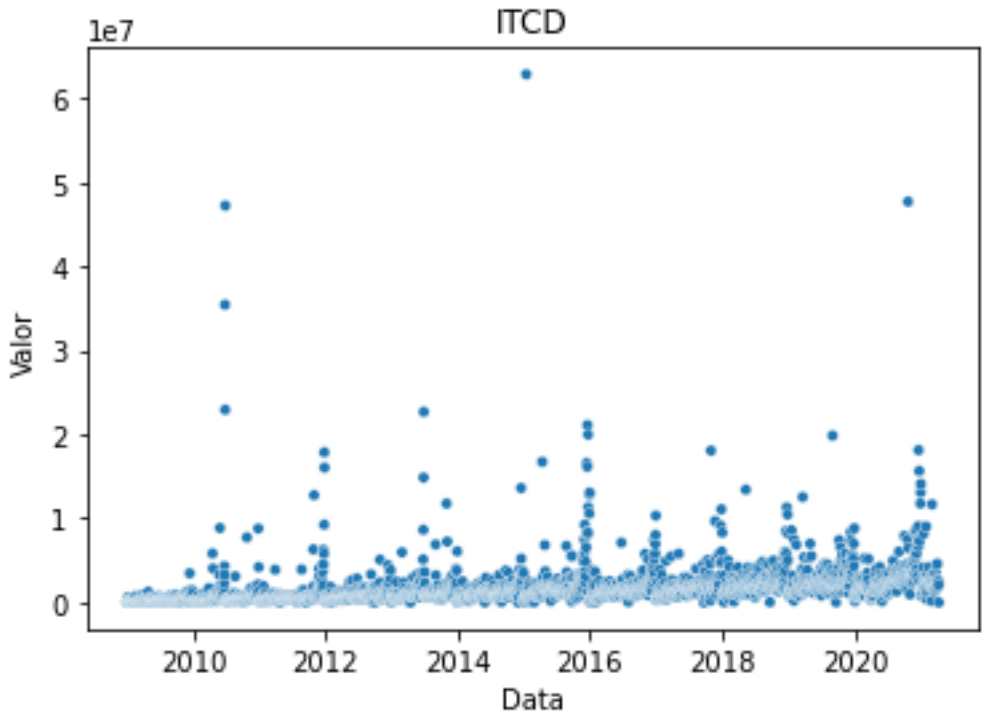

Figura 7 – Arrecadação nominal do ITCD do RS

Denota-se que o ICMS, além de possuir um maior valor médio nominal de arrecadação, é o tributo que possui maior variância (σ^2), o que certamente aumentará o valor do erro médio dos modelos que serão desenvolvidos. Outro aspecto a ser comentado é a existência de valores negativos, mesmo que em menor valor absoluto, nas séries. A existência de arrecadação negativa se deve ao fato de que dos valores computados já estão deduzidas as restituições e benefícios fiscais, sendo os meses com valores negativos, portanto, aqueles em que as restituições e/ou benefícios fiscais excederam a arrecadação bruta. Observando os gráficos, percebe-se visualmente que todas as séries possuem tendência de crescimento nominal, o que já era esperado por conta da inflação. Outra característica interessante é a sazonalidade da arrecadação do IPVA nos primeiros meses do ano devido aos pagamentos à vista e das primeiras parcelas. No ITCD percebem-se alguns picos de arrecadação, mas não fica visualmente claro se há alguma sazonalidade.

4.3. Os dados reais

Atualizando os valores a valor presente, fica observado o aumento da magnitude dos novos números. Como a variância é, como também dito na análise dos valores nominais, fator que impacta diretamente o desempenho do modelo, o aumento

desta medida, decorrido da atualização, aumentará o desafio para que os modelos prevejam a arrecadação com grau aceitável de erro.

Medida	ICMS	IPVA	ITCD
Quantidade	3.167,00	3.119,00	3.097,00
Média	176.546.450,23	15.716.591,77	2.767.067,95
Desvio Padrão (σ)	225.599.315,66	29.589.472,15	4.523.596,03
Mínimo	-209.539,42	-356,90	-3.004,87
25%	27.993.106,87	3.130.045,03	1.216.229,59
50%	68.630.690,51	8.435.535,40	2.003.726,62
75%	259.310.821,37	17.177.296,69	3.063.960,79
Máximo	1.560.701.121,06	421.701.485,08	113.094.527,20

Figura 8 – Descrição da arrecadação real dos tributos estaduais

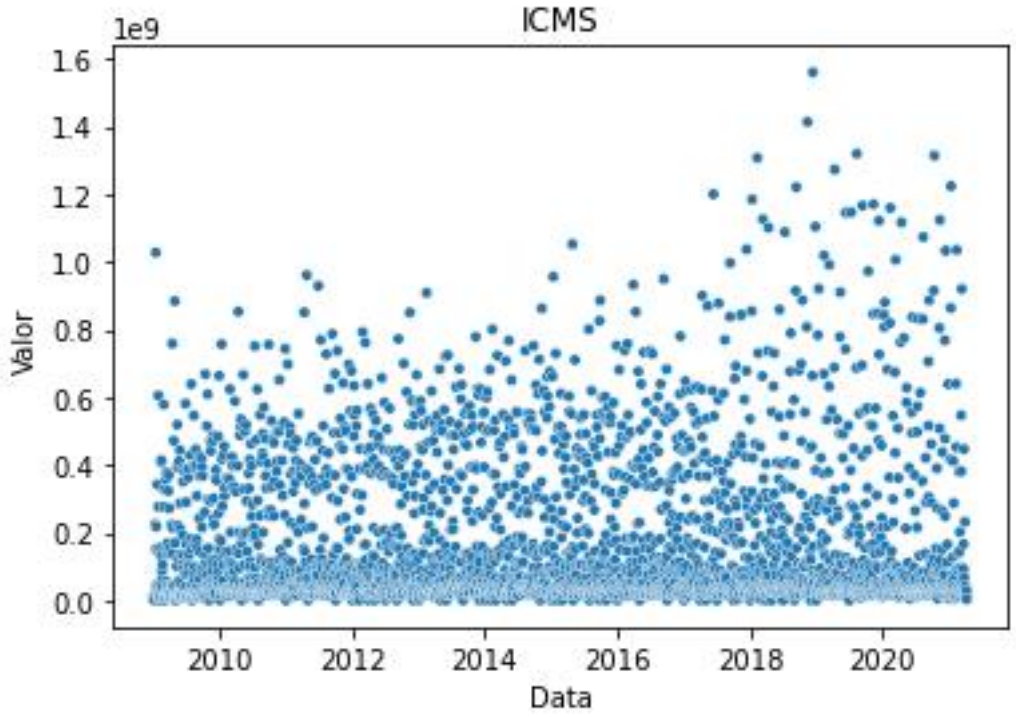

Figura 9 – Arrecadação real do ICMS do RS

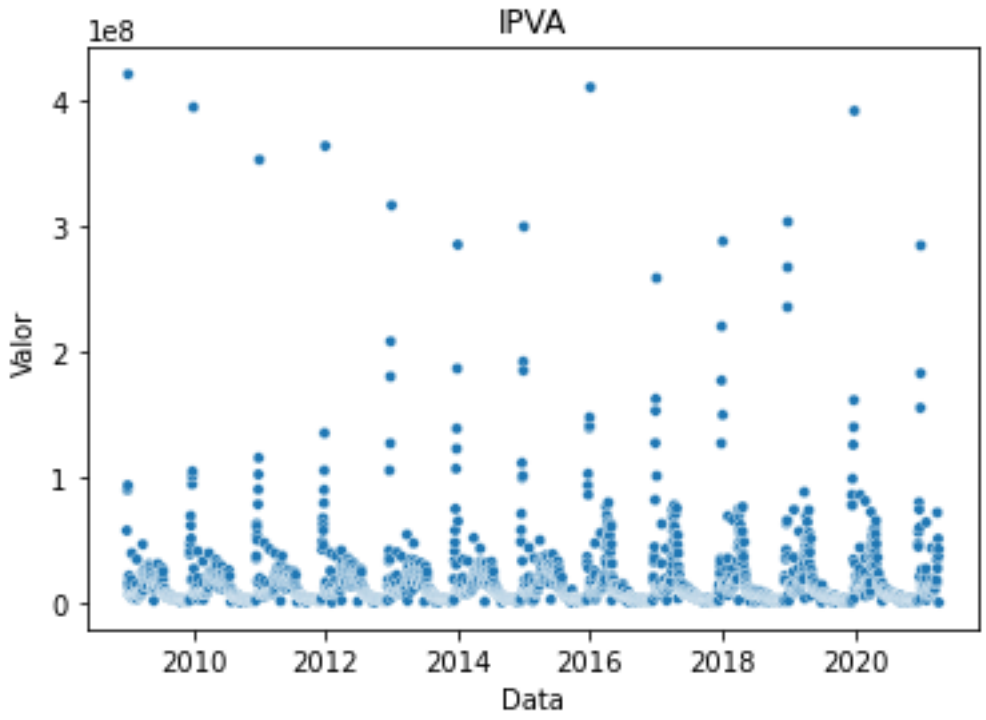

Figura 10 – Arrecadação real do IPVA do RS

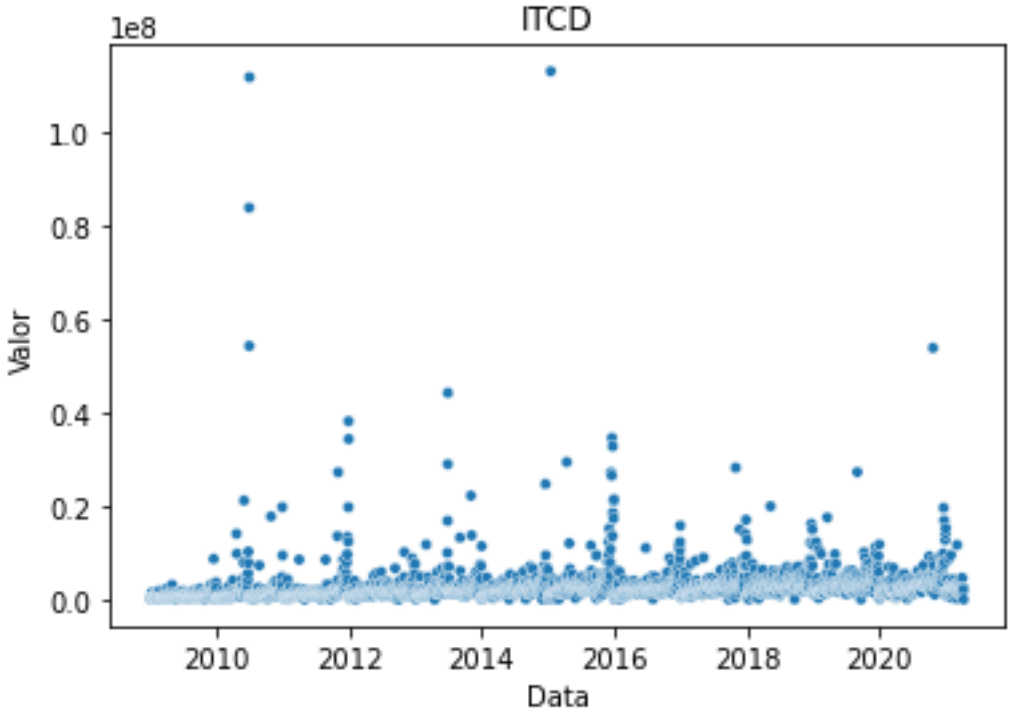

Figura 11 – Arrecadação real do ITCD do RS

Após a atualização dos valores, percebe-se que a tendência de aumento ficou reduzida em todas as séries quando em comparação com os valores nominais. Com a inflação eliminada dos dados, parte-se para uma análise mais aprofundada destes.

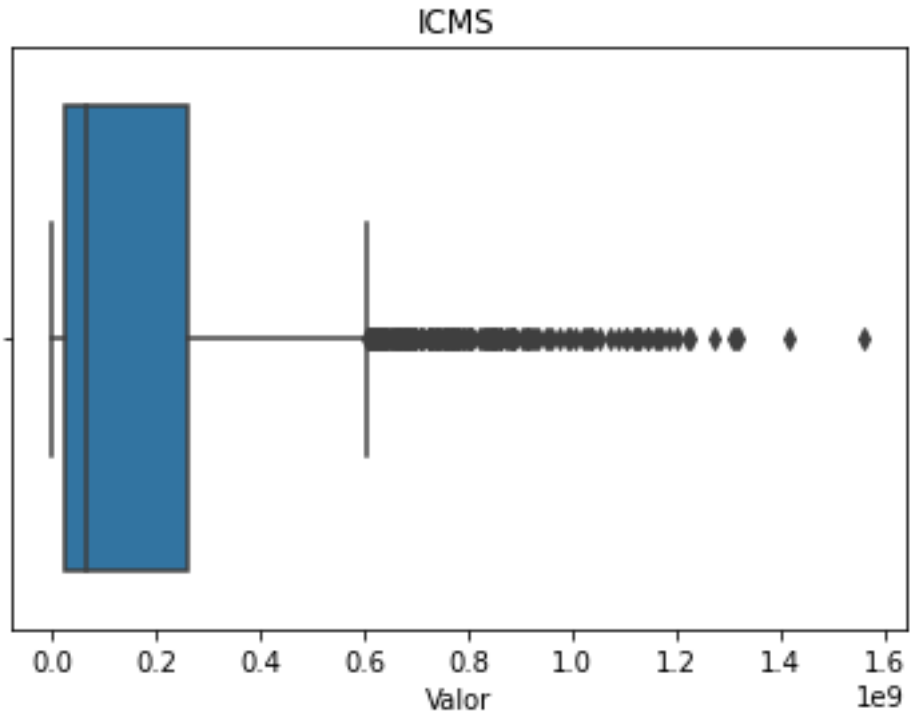

Figura 12 – Boxplot da arrecadação real do ICMS do RS

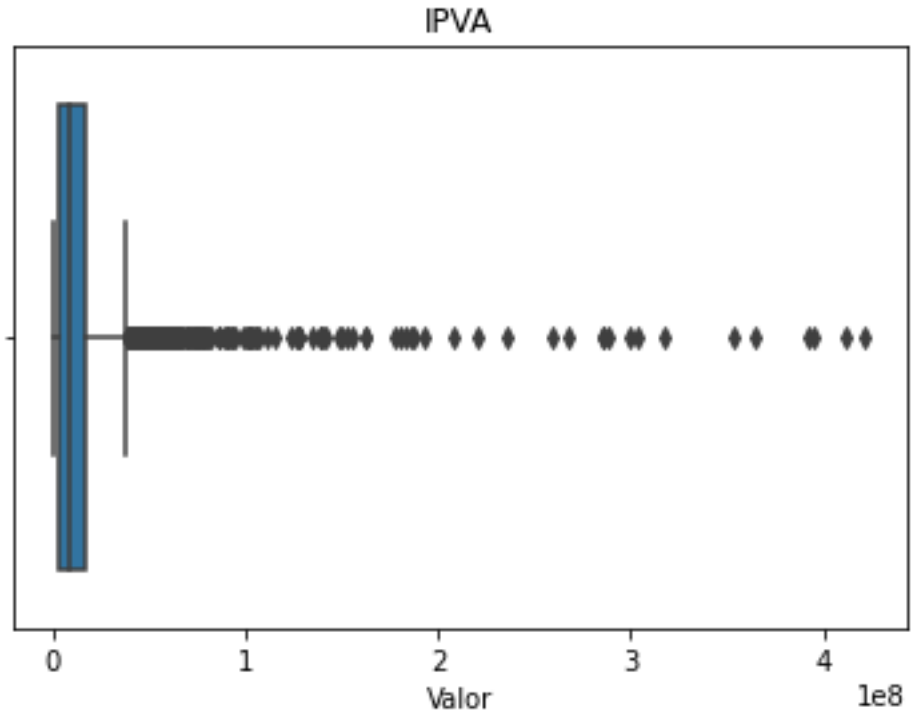

Figura 13 – Boxplot da arrecadação real do IPVA do RS

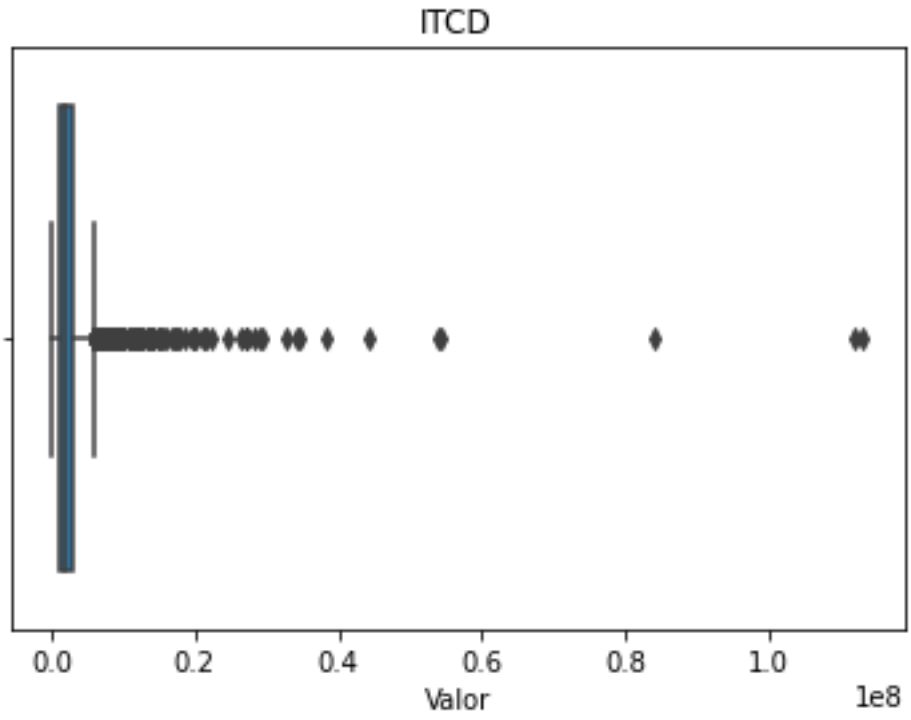

Figura 14 – Boxplot da arrecadação real do ITCD do RS

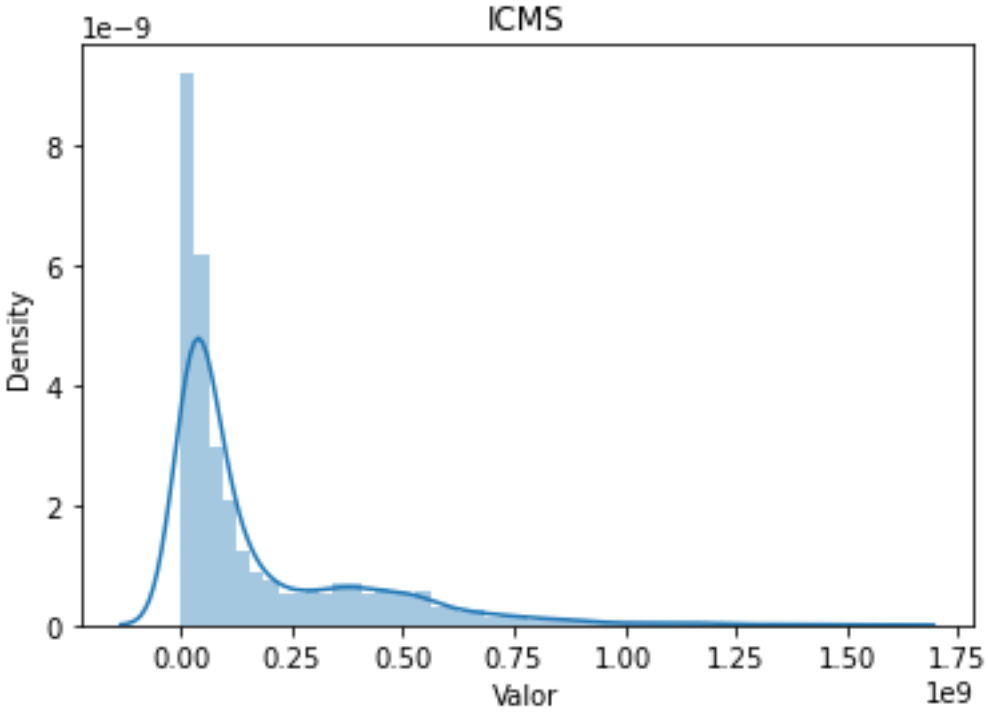

Figura 15 - Distribuição de probabilidades da arrecadação real do ICMS

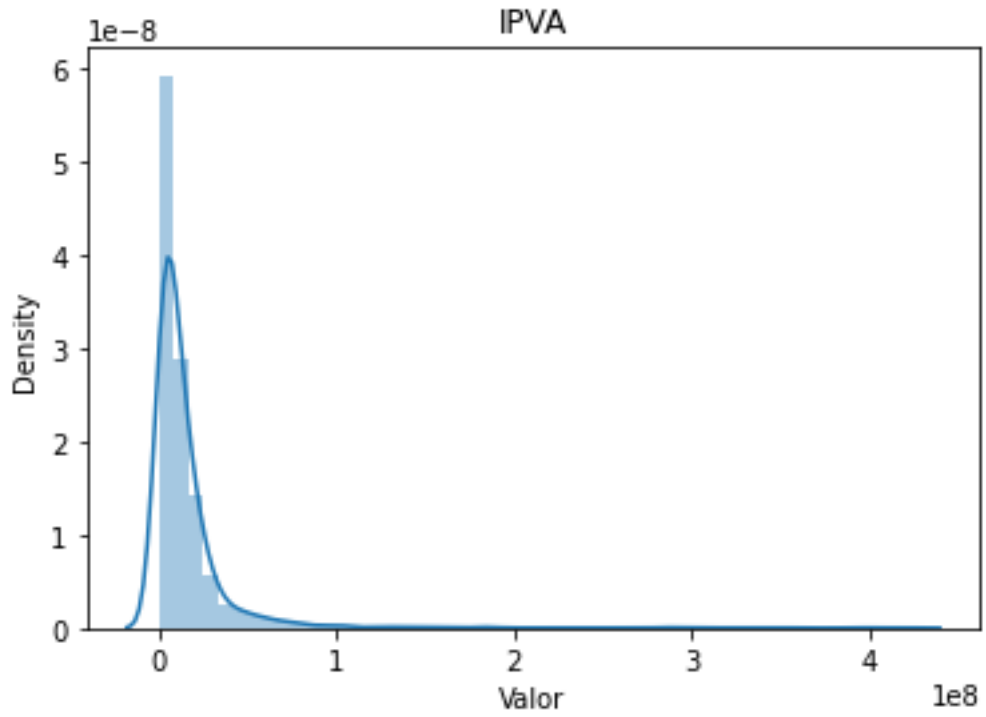

Figura 16 - Distribuição de probabilidades da arrecadação real do IPVA

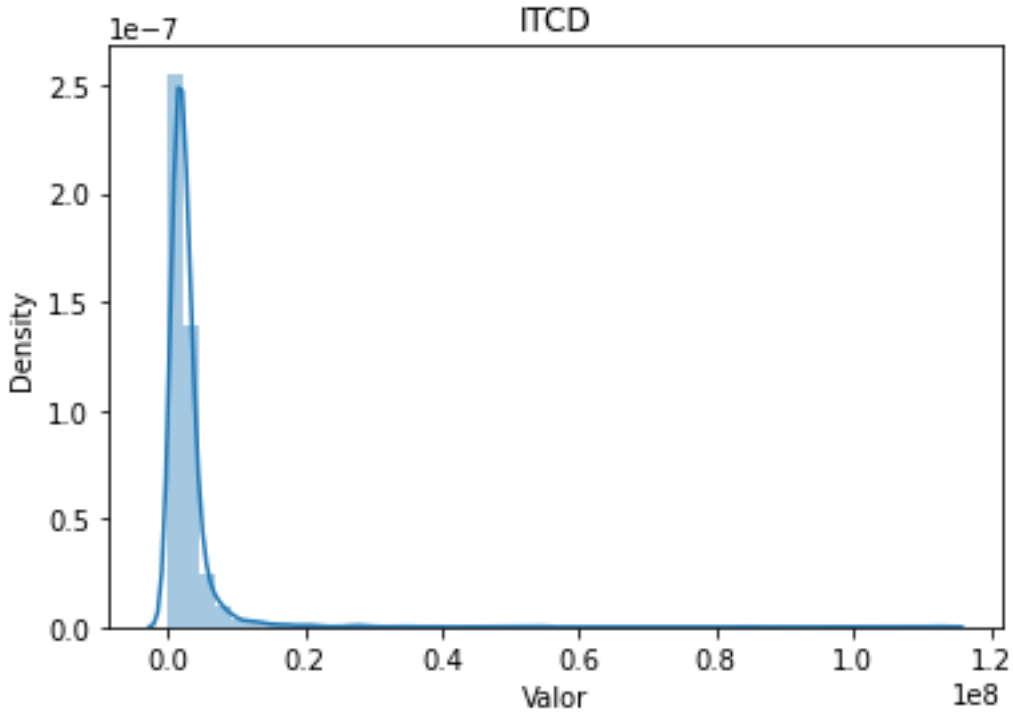

Figura 17 - Distribuição de probabilidades da arrecadação real do ITCD

Pela amplitude dos *boxlots* é possível verificar a variância das séries temporais da arrecadação dos tributos. O ICMS, com variância bem superior aos demais, é o que possui maior intervalo interquartílico, seguido pelo IPVA e ITCD, com menores variâncias. O mesmo ocorre com os *outliers*, estando presentes em maior quantidade no ICMS do que nos demais tributos. Sendo valores que têm grande potencial de

atrapalharem as predições, os *outliers* devem passar por tratamento para que suas importâncias sejam reduzidas. Além disso, cada modelo preditivo comporta-se de maneira distinta em relação aos tratamentos, devendo ser escolhido o melhor método para cada modelo. Neste trabalho, para os modelos utilizando redes neurais LSTM, os dados serão tratados com três *scalers* do pacote *scikit-learn*: *StandardScaler*, *RobustScaler e PowerTransformer* e mantido aquele que apresentar menor erro para cada modelo. Já para o Prophet o trabalho seguirá as orientações especificadas no *paper Forecasting at Scale*[4], que deu origem ao algoritmo, e as orientações da documentação do Prophet[5], removendo os outliers.

5. Modelos preditivos para a arrecadação

5.1.1. Facebook™ Prophet

O Facebook™ Prophet é uma biblioteca que permite a predição de séries temporais de forma acurada e rápida. Conforme a página da biblioteca[6], o Prophet é capaz de lidar com predição de séries temporais com tendências não-lineares, preenchendo-as com sazonalidade anual, semanal e/ou diária. Como observação, a página indica que o algoritmo performa melhor em séries temporais com fortes efeitos sazonais e várias “temporadas” de dados históricos. Sendo assim, espera-se que as predições da arrecadação do IPVA performem melhor que as do ICMS e ITCD, uma vez que a arrecadação daquele é visivelmente mais sazonal que as destes.

Na mesma página ainda é informado que modelos do Prophet lidam bem com *missing data* e *outliers*, além de lidar bem com mudanças de tendência. Com relação a estes pontos não há nos *dataframes* deste trabalho casos que exijam tratamento diferenciado de *missing values*, uma vez que dados ausentes puderam ser preenchidos com consultas a outras bases de dados. Com relação aos *outliers*, uma vez que o objetivo é buscar o menor erro no modelo, será adotado o procedimento descrito no final do item 4.3 (remoção de *outliers*) e também a comparação com modelo sem a remoção destes valores, verificando o que melhor performa.

5.1.2. Redes neurais Long Short-Term Memory (LSTM)

[4] Disponível em: <https://peerj.com/preprints/3190.pdf>. Acesso em 12 de abril de 2021.
[5] Disponível em: <https://facebook.github.io/prophet/docs/outliers.html>. Acesso em 12 de abril de 2021.
[6] Disponível em: <https://facebook.github.io/prophet/>. Acesso em 12 de abril de 2021.

Criadas em 1997 por Sepp Hochreiter e Jürgen Schmidhuber, as redes neurais LSTM permitem a predição de séries temporais utilizando neurônios de entrada de três dimensões (tamanho do lote – *batch size*, tamanho da janela temporal – *timesteps* e quantidade de *features* para cada janela temporal – *input dimension*), diferentemente das redes neurais *Multilayer Perceptron* (MLP) que necessitam apenas de *dataframes* com duas dimensões (*batch size* e *input dimension*). Constituem um tipo de modelo altamente sensível a *outliers* e *missing values*, devendo ser tratados adequadamente, como descrito ao final do item 4.3.

Neste trabalho a transformação dos *dataframes* em *dataframes* de três dimensões é realizada pelo método *cria_intervalos_temporais* da classe *LSTMUtil*, dentro do arquivo *ModelosUtil.py*.

```
@staticmethod
def cria_intervalos_temporais(np_array, n_intervalos=5):
    """ Dado um array NumPy com os valores diários, gera sequências
temporais com 3 dimensões
    para alimentarem a rede neural LSTM. """

    np_valores = np_array
    np_sequencia = np.empty((0, n_intervalos, 1))

    for i in range(n_intervalos, len(np_valores)):
        # Adiciona os itens que comporão uma sequência
        # Cada item é composto por uma sequência, n_intervalos intervalos
de tempo e 1 feature)
        np_item = np.empty((0, n_intervalos, 1))
        np_item = np.append(np_item, np_valores[(i-n_intervalos):i,
0].reshape(1, n_intervalos, 1), axis=0)
        # Adiciona uma sequência à lista de sequências
        np_sequencia = np.append(np_sequencia, np_item, axis=0)

    return np_sequencia
```

Figura 18 – Método de transformação de arrays NumPy de duas dimensões em arrays de três dimensões

Os modelos LSTM serão construídos com base na API Funcional (*Functional API*) do *Keras*, permitindo maior flexibilidade que a tradicional e mais simples API Sequencial (*Sequential API*), como, por exemplo, o uso de camadas *Embedding* para utilizar parâmetros das datas, como dia e mês, como inputs do modelo sem que sejam necessariamente tratados como *features* dentro das janelas temporais (uma vez que datas sempre são sequenciais, a adição de todas as datas em todos os *timesteps* tornaria o modelo mais lento sem acrescentar poder preditivo). Sendo assim, a API Funcional permite a integração entre vários tipos de camadas e até mesmo dimensões em um único modelo.

5.2. Modelos com apenas uma variável quantitativa

Os modelos com apenas uma variável quantitativa serão alimentados apenas com os valores das datas e da arrecadação, ou seja, buscarão realizar as predições utilizando somente a autocorrelação e sazonalidade como fatores preditivos.

5.2.1. Prophet

5.2.1.1. Utilizando o Prophet com remoção de *outliers*

Os *dataframes* de treino e teste foram divididos na proporção de 80% e 20%, respectivamente, do *dataframe* inicial. No primeiro momento serão retirados o *dataframe* de treino os *outliers*, sendo considerados como *outliers* aqueles valores que estiverem acima de 1,5 desvio quartil acima do terceiro quartil e abaixo de 1,5 desvio quartil abaixo do primeiro quartil. O método *transforma_dataframe* da classe *ProphetUtil* é responsável por converter o *dataframe* inicial em um *dataframe* no formato aceito pelo Prophet, com uma coluna com o nome “ds”, contendo as datas, e outra com o nome “y”, contendo os valores.

```
@staticmethod
def transforma_dataframe(df, list_colunas):
    """ Dado um dataframe e uma lista com as colunas de data e e valor
(nesta ordem), retorna um dataframe
    para que possa ser feito o 'fit' no Prophet. """
    pd_prophet = pd.DataFrame(df).loc[:, list_colunas]
    pd_prophet[list_colunas[0]] = to_datetime(pd_prophet[list_colunas[0]])

    return pd_prophet.rename(columns={list_colunas[0]: 'ds',
list_colunas[1]: 'y'}).reset_index(drop=True)
```

Figura 19 – Método de transformação do dataframe em um dataframe do Prophet

	ds	y
0	2009-01-02	1.081559e+05
1	2009-01-05	2.388174e+05
2	2009-01-06	3.820471e+05
3	2009-01-07	2.925644e+05
4	2009-01-09	1.486864e+06
	...	...
3092	2021-04-01	1.896179e+06
3093	2021-04-05	2.368007e+06
3094	2021-04-06	2.492219e+06
3095	2021-04-07	2.101995e+06
3096	2021-04-08	3.600000e+02

Figura 20 – Exemplo de formato do dataframe de entrada do Prophet

O treinamento do modelo é realizado pelo método *fit* da classe *Prophet* e a predição é realizada pelo método *predict* da mesma classe. Retornadas as predições para cada tributo, são gravados os erros e as datas dos *dataframes* de testes e plotados os gráficos com os resultados.

```
# Predição com a remoção de 'outliers'
for tributo in pd_arrecad_diaria['Tributo'].unique():
    # Calcula os valores em termos absolutos
    prophet = Prophet(daily_seasonality=True)

    pd_prophet = ProphetUtil.transforma_dataframe(arrecad_diaria[tributo],
['Data', 'Valor'])
    df_treino, df_teste = ProphetUtil.divide_treino_teste(pd_prophet)

    # Remove os 'outliers' do dataframe de treino
    primeiro_quartil = df_treino['y'].quantile(.25)
    terceiro_quartil = df_treino['y'].quantile(.75)
    desvio_quartil = (terceiro_quartil-primeiro_quartil)/2
    df_treino =
df_treino[(df_treino['y']<terceiro_quartil+1.5*desvio_quartil) &
(df_treino['y']>primeiro_quartil-1.5*desvio_quartil)]

    df_teste.reset_index(drop=True, inplace=True)
    prophet.fit(df_treino)
    predito = prophet.predict(pd.DataFrame(df_teste['ds']))

    # Grava os erros
    rmse = mean_squared_error(pd.DataFrame(df_teste['y']).values,
predito['yhat'].values) ** (1 / 2)
    mae = mean_absolute_error(pd.DataFrame(df_teste['y']).values,
predito['yhat'].values)

    # Plota as predições
    fig, (sub1) = plt.subplots(1, 1, sharex=True)
    sub1.fill_between(df_teste['ds'], predito['yhat_upper'],
predito['yhat_lower'], facecolor='dodgerblue')
    pred, = plt.plot(df_teste['ds'], predito['yhat'], c='blue',
label='Predito')
    pred_sup, = plt.plot(df_teste['ds'], predito['yhat_upper'],
c='royalblue')
    pred_inf, = plt.plot(df_teste['ds'], predito['yhat_lower'],
c='royalblue')
    real = plt.scatter(df_teste['ds'], df_teste['y'], s=3, c='orange')
    plt.legend([pred, pred_sup, real],
               ['Predito', 'Predito (limites superior e inferior)',
'Real'],
               fontsize=8)
    fig.autofmt_xdate()
    plt.xlabel('Data')
    plt.ylabel('Valor (R$)')
    plt.title(tributo)
    plt.show()

    pd_datas_testes.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'Inicio'] = df_teste.reset_index().loc[0, 'ds']
    pd_datas_testes.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'Fim'] = df_teste.reset_index().loc[len(df_teste) - 1, 'ds']
    pd_datas_treinos.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'Inicio'] = df_treino.reset_index().loc[0, 'ds']
    pd_datas_treinos.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'Fim'] = df_treino.reset_index().loc[len(df_treino) - 1, 'ds']
    pd_performance.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'MAE'] = mae
    pd_performance.loc[tributo+' - Prophet - Univariável - Com Remoção de
Outliers', 'RMSE'] = rmse
    pd_stats.loc[tributo+' - DP', 'dp'] = df_teste['ds'].std()

    # Plota os componentes
    prophet.plot_components(predito)
```

Figura 21 – Trecho do código em que é feita a predição pelo Prophet com a remoção de outliers

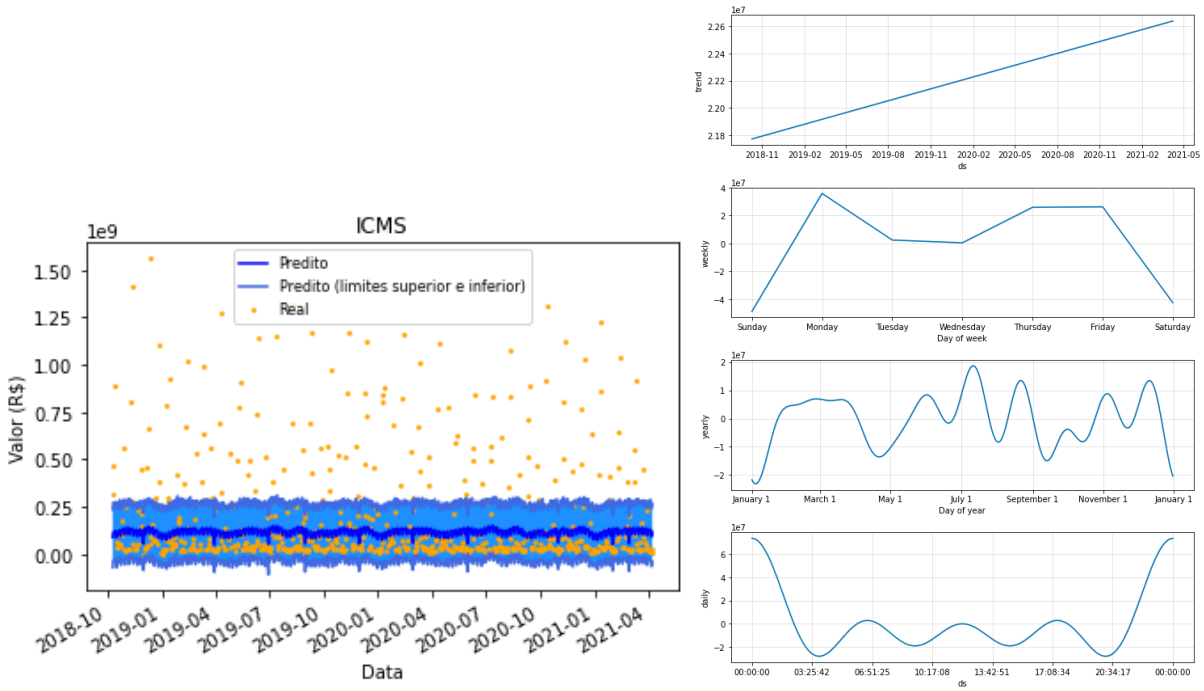

Figura 22 – Predição da arrecadação do ICMS (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), com remoção de outliers

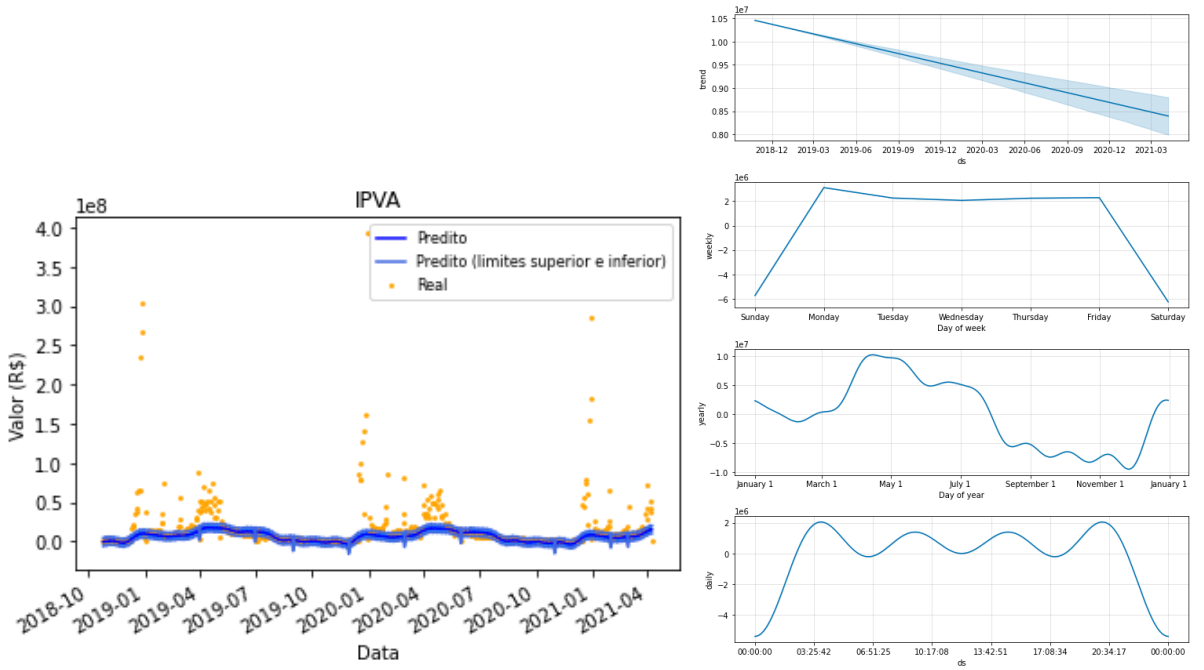

Figura 23 – Predição da arrecadação do IPVA (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), com remoção de outliers

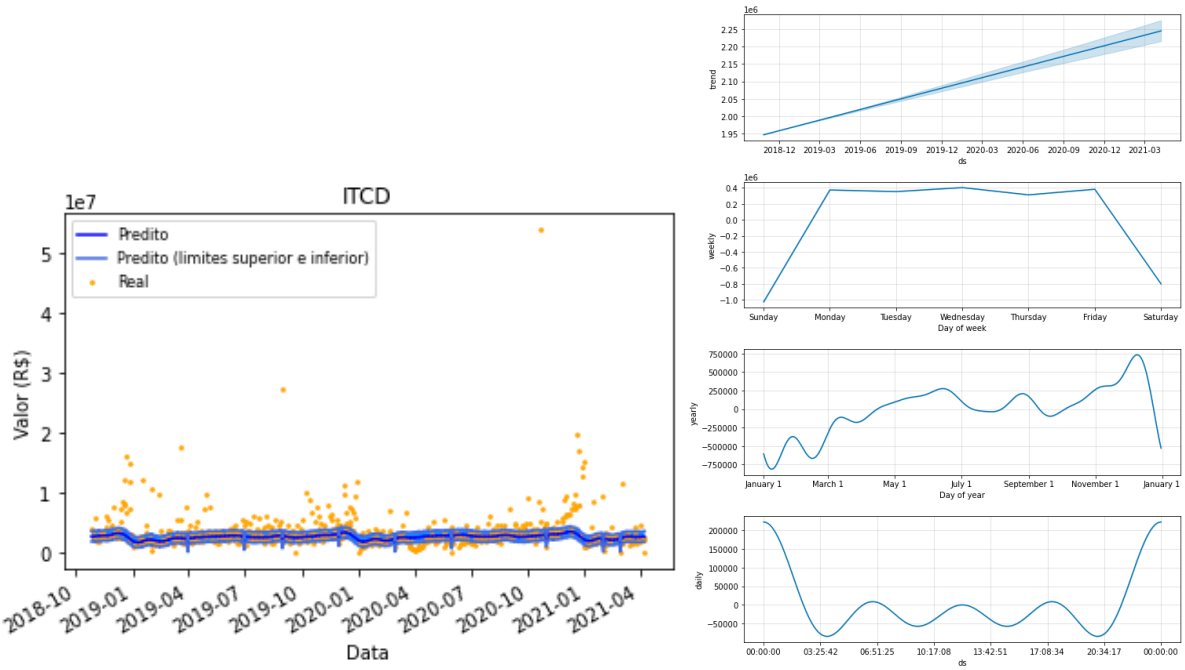

Figura 24 – Predição da arrecadação do ITCD (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), com remoção de outliers

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	285.511.789,84	277.909.138,33	1,027357
IPVA	33.424.786,01	33.042.123,48	1,011581
ITCD	3.388.334,97	3.311.754,46	1,023124

Figura 25 – Root Mean Squared Error dos resíduos das predições do Prophet, com exclusão de outliers

Com a exclusão dos *outliers* do set de treinamento, percebe-se que os modelos preditivos para os três tributos permanecem praticamente inalterados quando da ocorrência de *outliers* nos *datasets* de testes. Quanto aos componentes das predições, verifica-se que, com exceção ao IPVA, há uma tendência de crescimento. A sazonalidade semanal é praticamente inexistente para os três tributos, enquanto a sazonalidade anual fica clara principalmente para o IPVA na primeira metade do ano e fica maior para o ITCD à medida que o final do ano se aproxima. Já para o ICMS não há uma sazonalidade muito clara, alternando entre topos e fundos no decorrer do ano. Quanto aos resíduos, todos os tributos tiveram predições com a razão entre a raiz quadrada do erro quadrático médio (A) pelo desvio padrão do set de testes (B) maior que 1, o que é pouco satisfatório para fins preditivos, mas considerando um modelo que apenas avalia a sazonalidade, sem relacionar outros fatores, possui um bom desempenho. O uso da métrica C, descrita como a razão de A e B visa avaliar os modelos sem penalizá-los nas predições de séries temporais com variância elevada.

5.2.1.2. Utilizando o Prophet sem remoção de *outliers*

Para fins comparativos, serão executados os mesmos procedimentos do item 5.2.1.1, com exceção da exclusão dos *outliers*, mantendo-se, portanto, todos os dados no *dataset* de treinamento, deixando o Prophet tratar os *outliers* automaticamente.

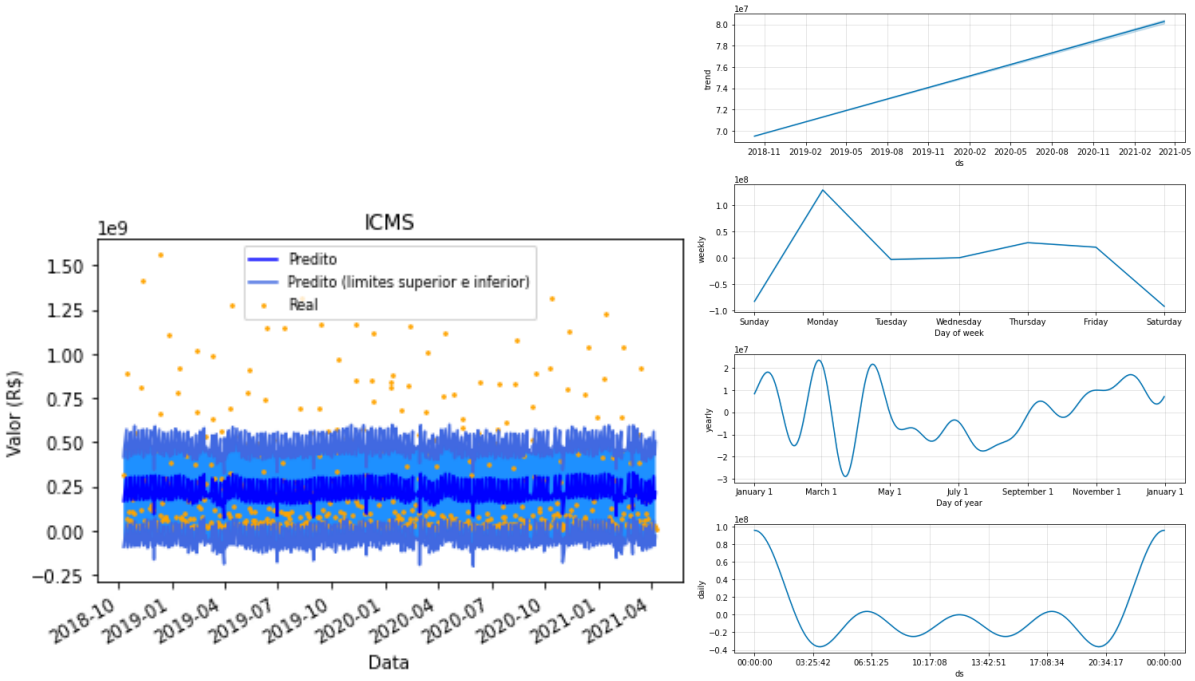

Figura 26 – Predição da arrecadação do ICMS (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), sem remoção de outliers

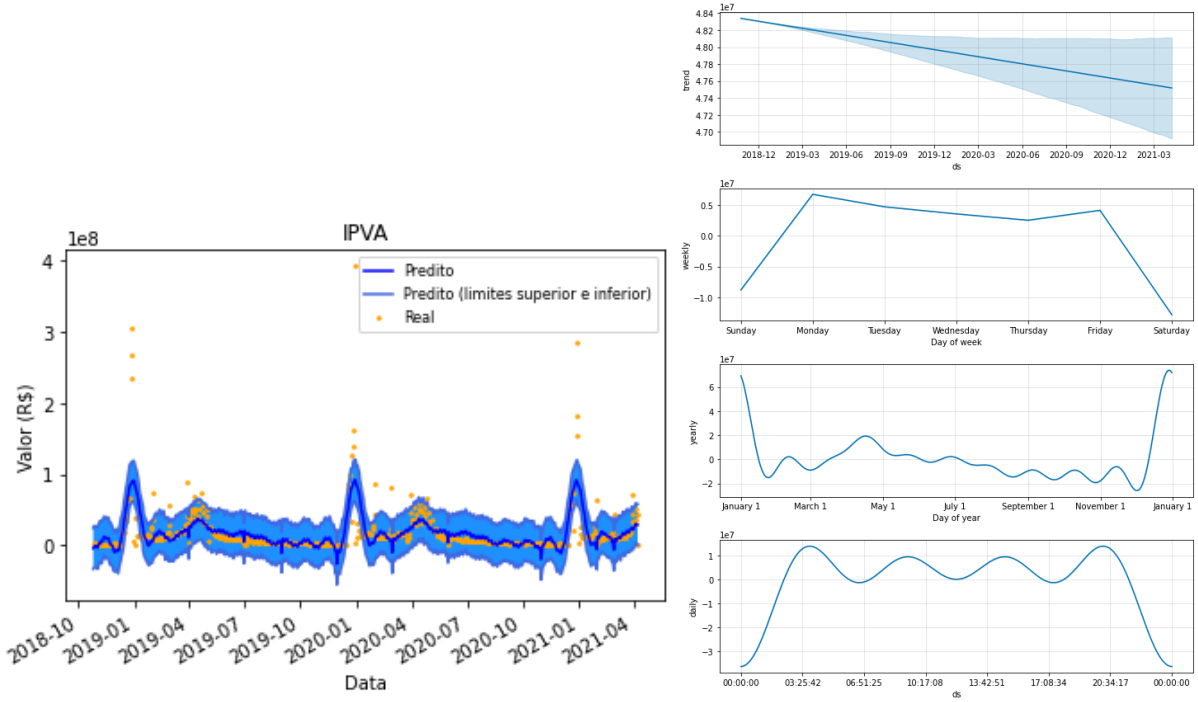

Figura 27 – Predição da arrecadação do IPVA (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), sem remoção de outliers

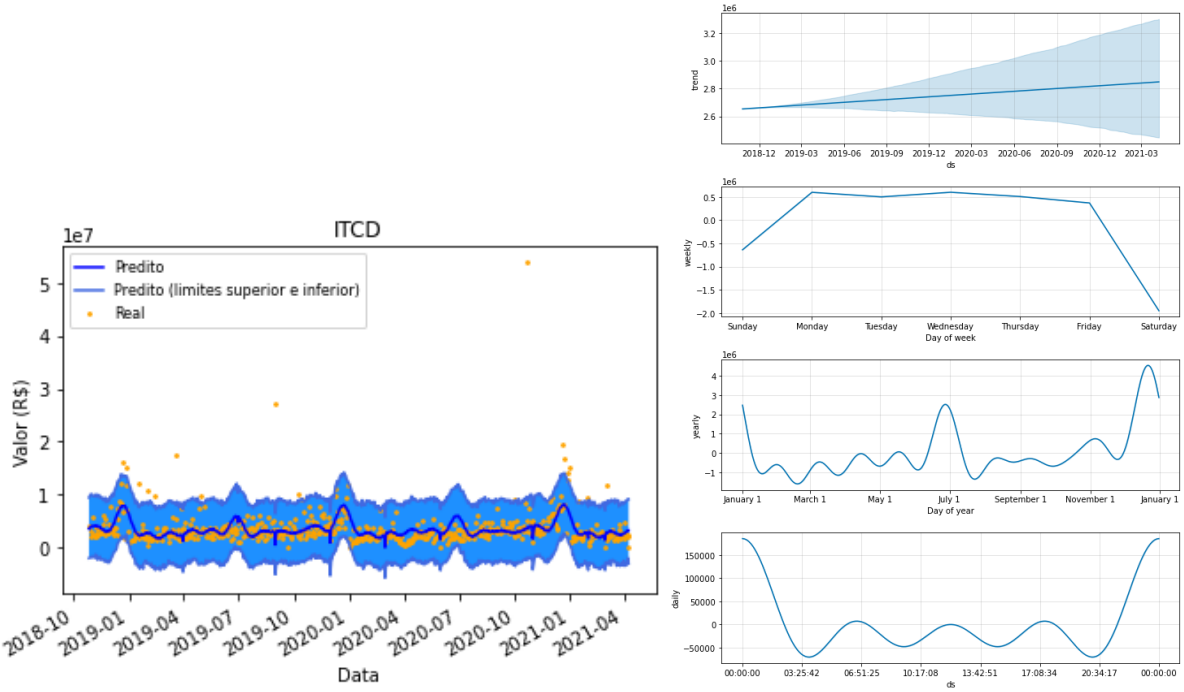

Figura 28 – Predição da arrecadação do ITCD (à esquerda) e componentes da predição (de cima para baixo: tendência, sazonalidade semanal, anual e diária), sem remoção de outliers

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	274.267.673,11	277.909.138,33	0,986897
IPVA	26.008.612,76	33.042.123,48	0,787135
ITCD	3.175.062,75	3.311.754,46	0,958725

Figura 29 – Root Mean Squared Error dos resíduos das predições do Prophet, sem exclusão de outliers

Ao realizar o treinamento sem excluir os *outliers*, verifica-se uma melhora na performance de todos os modelos, principalmente no IPVA, que teve uma redução de quase 22%. Conclui-se, portanto que a remoção dos outliers para as séries temporais destes três tributos é prejudicial às predições. Percebe-se que quanto mais sazonal é a série temporal, maior é o impacto da exclusão dos *outliers* nos erros das predições, muito provavelmente porque os *outliers* também possuem sazonalidade e sua remoção faz com que os modelos a ignorem. Comparando a figura 19 com a 22 e a 20 com a 23 é possível verificar claramente o potencial que os valores removidos dão aos modelos dos tributos com maior sazonalidade, que deixar de prever a arrecadação futura como uma linha quase horizontal (figuras 19 e 20) e passam a apresentar curvas mais condizentes com os valores reais (figuras 22 e 23). É possível visualizar também que nos modelos sem a exclusão de valores um maior número de pontos representativos dos valores reais se encontra dentro da área azul claro (margem de erro), comprovando mais uma vez a melhora da performance do modelo.

Com relação à tendência todos os modelos mantiveram previsões semelhantes às dos modelos com remoção de *outliers*. Com relação à sazonalidade, o ICMS permaneceu sem uma sazonalidade visível. Já o IPVA teve sua sazonalidade nos primeiros meses do ano, principalmente em janeiro, majorada por conta dos pagamentos à vista e das primeiras parcelas do tributo. Com relação ao ITCD ficou clara a sazonalidade do tributo nos meses de junho e janeiro, diferentemente do que apontava o modelo com remoção de outliers.

5.2.2. Rede Neural LSTM

5.2.2.1. Padronização dos dados

Diferentemente do Prophet, os dados inseridos nos modelos LSTM necessitam, além do adequado tratamento de *outliers*, de padronização, o que além de reduzir eventuais anomalias nos *datasets*, permite que os pesos sejam ajustados mais rapidamente durante o *backpropagation*[7].

> "Os dados para seu problema de predição de sequências provavelmente precisam ser escalados quando do treinamento de uma rede neural, tal como a rede neural recorrente LSTM. Quando uma rede neural é preenchida com dados não escalados que possuem um grande intervalo de valores (Ex.: quantidades em dezenas ou centenas) é possível que entradas com grandes valores reduzam a velocidade de aprendizado e a convergência da sua rede neural e, em alguns casos, não permite que a rede efetivamente aprenda o modelo". (Brownlee, Jason. *Long Short-Term Memory Networks With Python*. Machine Learning Mastery, 2017, p. 27, livre tradução)[8]

Neste trabalho, como explicitado no item 4.3, serão utilizados três *scalers* do pacote *sklearn.preprocessing*: *StandardScaler*, *RobustScaler* e *PowerTransformer (método yeo-johnson)* e comparados os resultados no *set* de testes (*out-of-sample*, sendo 80% do *set* original utilizado para treinamento e 20% para testes), mediante treinamento com apenas 100 *epochs*, sendo utilizado no treinamento do modelo final aquele que obtiver menor erro. O *PowerTransformer* foi utilizado com o método *yeo-johnson* uma vez que o outro método, *box-cox*, só funciona com valores positivos e, como foi visto no item 4.2, há valores negativos na base de dados, inviabilizando seu

[7] Algoritmo que ajusta os pesos atribuídos aos neurônios com o objetivo de reduzir o erro final do modelo. (Brownlee, Jason. *Long Short-Term Memory Networks With Python*. Machine Learning Mastery, 2017, p. 19)

[8] No original: The data for your sequence prediction problem probably needs to be scaled when training a neural network, such as a Long Short-Term Memory recurrent neural network. When a network is fit on unscaled data that has a range of values (e.g. quantities in the 10s to 100s) it is possible for large inputs to slow down the learning and convergence of your network, and in some cases prevent the network from effectively learning your problem.

uso. O *StandardScaler* é um padronizador conhecido na estatística em que é subtraído de cada valor a média e o valor resultante dividido pelo desvio padrão:

$$valor\_padronizado = \frac{x - \bar{x}}{\sigma}$$

Já o *RobustScaler* procura tratar melhor os *outliers*, subtraindo de cada valor a mediana e dividindo o valor resultante pelo intervalo interquartil (diferença entre o terceiro e primeiro quartis).

$$valor\_padronizado = \frac{x - mediana}{Q_3 - Q_1}$$

Por sua vez, a transformação yeo-johnson do *PowerTransformer* é bem mais complexa, aplicando uma série de funções condicionais para resultar no valor transformado. O valor λ é determinado através da máxima verossimilhança estatística (maximum-likelihood estimation – MLE).

$$valor\_padronizado = \begin{cases} \{(x+1)^{\lambda} - 1\}/\lambda & (x \geq 0, \lambda \neq 0), \\ \log(x+1) & (x \geq 0, \lambda = 0), \\ -\{(-x+1)^{2-\lambda} - 1\}/(2-\lambda) & (x < 0, \lambda \neq 2), \\ -\log(-x+1) & (x < 0, \lambda = 2). \end{cases}$$

Quanto à estrutura da rede neural, apesar de já ter sido utilizada neste comparativo, deixarei para comentá-la mais à frente, dado que merece maior destaque e detalhamento.

```
# Cria modelo com única variável quantitativa utilizando LSTM
from src.ModelosUtil import LSTMUtil
from src.ModelosNN import LSTMUnivariada
import tensorflow.keras.optimizers as ko
from tensorflow import keras
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
TensorBoard

comparativo = pd.DataFrame(columns=['StandardScaler', 'RobustScaler',
'PowerTransformer'])

for tributo in pd_arrecad_diaria['Tributo'].unique():
    # Utiliza o mesmo método do Prophet para tornar os resultados
comparáveis
    df_treino, df_teste =
ProphetUtil.divide_treino_teste(arrecad_diaria[tributo])
    print('Tributo ' + tributo + ' - Início DF teste : ' + str(
        df_teste.reset_index().loc[0, 'Data']) + ' Fim DF teste : ' + str(
```

```
        df_teste.reset_index().loc[len(df_teste) - 1, 'Data']))
    df_treino = LSTMUtil.transforma_dataframe(df_treino, 'Data')
    df_teste = LSTMUtil.transforma_dataframe(df_teste, 'Data')

    # Faz o Label Encoder do dia, dia da semana e mês (apesar de dia e mês
serem numéricos, o Label Encoder inicia a contagem em 0 ao invés de 1)
    encoder_dia = LabelEncoder()
    dia_treino_enc = encoder_dia.fit_transform(df_treino['Dia'].values)
    dia_teste_enc = encoder_dia.transform(df_teste['Dia'].values)
    encoder_mes = LabelEncoder()
    mes_treino_enc = encoder_mes.fit_transform(df_treino['Mes'].values)
    mes_teste_enc = encoder_mes.transform(df_teste['Mes'].values)
    encoder_dia_semana = LabelEncoder()
    dia_semana_treino_enc =
encoder_dia_semana.fit_transform(df_treino['Dia_Semana'].values)
    dia_semana_teste_enc =
encoder_dia_semana.transform(df_teste['Dia_Semana'].values)
    encoder_dia_semana = LabelEncoder()
    dia_semana_treino_enc =
encoder_dia_semana.fit_transform(df_treino['Dia_Semana'].values)
    dia_semana_teste_enc =
encoder_dia_semana.transform(df_teste['Dia_Semana'].values)

    np_dia_mes_treino = np.concatenate((dia_treino_enc.reshape(-1, 1),
mes_treino_enc.reshape(-1, 1), dia_semana_treino_enc.reshape(-1, 1)),
axis=1)[5:]
    np_dia_mes_teste = np.concatenate((dia_teste_enc.reshape(-1, 1),
mes_teste_enc.reshape(-1, 1), dia_semana_teste_enc.reshape(-1, 1)),
axis=1)[5:]

    # Faz os testes com diversos "scalers" para verificar o com menor erro

    # Standard Scaler
    std_scaler = StandardScaler()
    valor_treino_std =
std_scaler.fit_transform(df_treino['Valor'].values.reshape(-1, 1))
    valor_teste_std =
std_scaler.transform(df_teste['Valor'].values.reshape(-1, 1))

    # A saída (label) é a arrecadação do dia seguinte ao último dia da
sequência
    saida_treino = valor_treino_std[5:]
    saida_teste = valor_teste_std[5:]

    valor_arrecadacao_serie_temporal_lstm_treino =
LSTMUtil.cria_intervalos_temporais(valor_treino_std)
    valor_arrecadacao_serie_temporal_lstm_teste =
LSTMUtil.cria_intervalos_temporais(valor_teste_std)

    model = LSTMUnivariada(df_treino)
    checkpoint =
ModelCheckpoint('checkpoint_regressor_'+tributo+'_teste_standard_scaler.hdf
5', monitor='loss', verbose=2,
                                save_best_only=True,
save_weights_only=False,
                                mode='auto', period=1)
    model.compile(optimizer=ko.Adam(lr=0.1), loss='mse')
    model.fit([np_dia_mes_treino,
valor_arrecadacao_serie_temporal_lstm_treino], saida_treino,
validation_data=([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste], saida_teste),
              epochs=100, batch_size=50, callbacks=[checkpoint])

    # Carrega o melhor modelo salvo pelo Checkpoint
```

```
model.load_weights('checkpoint_regressor_'+tributo+'_teste_standard_scaler.
hdf5')

    std_pred = model.predict([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste])
    mae_std =
mean_absolute_error(std_scaler.inverse_transform(saida_teste),
std_scaler.inverse_transform(std_pred))
    print('O MAE para o tributo '+tributo+' usando o "Standard Scaler" foi
de '+str(mae_std))

    comparativo.loc[tributo, 'StandardScaler'] = mae_std

    # Robust Scaler
    rbt_scaler = RobustScaler()
    valor_treino_rbt =
rbt_scaler.fit_transform(df_treino['Valor'].values.reshape(-1, 1))
    valor_teste_rbt =
rbt_scaler.transform(df_teste['Valor'].values.reshape(-1, 1))

    # A saída (label) é a arrecadação do dia seguinte ao último dia da
sequência
    saida_treino = valor_treino_rbt[5:]
    saida_teste = valor_teste_rbt[5:]

    valor_arrecadacao_serie_temporal_lstm_treino =
LSTMUtil.cria_intervalos_temporais(valor_treino_rbt)
    valor_arrecadacao_serie_temporal_lstm_teste =
LSTMUtil.cria_intervalos_temporais(valor_teste_rbt)

    model = LSTMUnivariada(df_treino)
    checkpoint =
ModelCheckpoint('checkpoint_regressor_'+tributo+'_teste_robust_scaler.hdf5'
, monitor='loss', verbose=2,
                                save_best_only=True,
save_weights_only=False,
                                mode='auto', period=1)
    model.compile(optimizer=ko.Adam(lr=0.1), loss='mse')
    model.fit([np_dia_mes_treino,
valor_arrecadacao_serie_temporal_lstm_treino], saida_treino,
validation_data=([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste], saida_teste),
              epochs=100, batch_size=50, callbacks=[checkpoint])

    # Carrega o melhor modelo salvo pelo Checkpoint

model.load_weights('checkpoint_regressor_'+tributo+'_teste_robust_scaler.hd
f5')

    rbt_pred = model.predict([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste])
    mae_rbt =
mean_absolute_error(rbt_scaler.inverse_transform(saida_teste),
rbt_scaler.inverse_transform(rbt_pred))
    print('O MAE para o tributo '+tributo+' usando o "Robust Scaler" foi de
'+str(mae_rbt))

    comparativo.loc[tributo, 'RobustScaler'] = mae_rbt

    # Power Transformer (yeo-johnson)
    pwr_scaler = PowerTransformer()
    valor_treino_pwr =
pwr_scaler.fit_transform(df_treino['Valor'].values.reshape(-1, 1))
```

```
    valor_teste_pwr =
pwr_scaler.transform(df_teste['Valor'].values.reshape(-1, 1))

    # A saída (label) é a arrecadação do dia seguinte ao último dia da
sequência
    saida_treino = valor_treino_pwr[5:]
    saida_teste = valor_teste_pwr[5:]

    valor_arrecadacao_serie_temporal_lstm_treino =
LSTMUtil.cria_intervalos_temporais(valor_treino_pwr)
    valor_arrecadacao_serie_temporal_lstm_teste =
LSTMUtil.cria_intervalos_temporais(valor_teste_pwr)

    model = LSTMUnivariada(df_treino)
    checkpoint =
ModelCheckpoint('checkpoint_regressor_'+tributo+'_teste_power_transformer.h
df5', monitor='loss', verbose=2,
                                save_best_only=True,
save_weights_only=False,
                                mode='auto', period=1)
    model.compile(optimizer=ko.Adam(lr=0.1), loss='mse')
    model.fit([np_dia_mes_treino,
valor_arrecadacao_serie_temporal_lstm_treino], saida_treino,
validation_data=([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste], saida_teste),
              epochs=100, batch_size=50, callbacks=[checkpoint])

    # Carrega o melhor modelo salvo pelo Checkpoint

model.load_weights('checkpoint_regressor_'+tributo+'_teste_power_transforme
r.hdf5')

    pwr_pred = model.predict([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste])
    mae_pwr =
mean_absolute_error(pwr_scaler.inverse_transform(saida_teste),
pwr_scaler.inverse_transform(pwr_pred))
    print('O MAE para o tributo '+tributo+' usando o "Power Transformer"
foi de '+str(mae_pwr))

    comparativo.loc[tributo, 'PowerTransformer'] = mae_pwr
```

Figura 30 – Trecho do código em que são comparados os diversos scalers para normalização dos dados para a rede neural LSTM

Tributo	StandardScaler	RobustScaler	PowerTransformer
ICMS	115.762.908,93	199.449.107,75	115.261.123,26
IPVA	10.102.122,66	8.472.645,06	7.935.259,68
ITCD	2.085.123,60	2.085.306,60	1.760.417,32

Figura 31 – Comparativo do Mean Absolut Error (Erro Médio Absoluto) – MAE entre os scalers para cada tributo

Como resultado dos testes, foi elaborado o comparativo acima utilizando o Erro Médio Absoluto como parâmetro. Pela sua leitura é evidente que o PowerTransformer foi mais eficaz na redução dos erros dos modelos para os três tributos, motivo pelo

qual será o método escolhido para a padronização dos dados de entrada das redes neurais.

5.2.2.2. Preparação dos dados

Inicialmente separa-se os *dataframes* da arrecadação de cada tributo em *datasets* de treino e teste na proporção de 80% e 20%, respectivamente. Logo após extrai-se da data no formato *aaaa-MM-dd* os valores referentes a dia, dia da semana, mês e ano, criando novas colunas com estas features. Em seguida cada coluna desta é codificada com o *LabelEncoder*, convertendo-as em valores numéricos de 0 a *n*-1, sendo *n* a quantidade de valores distintos contidos em cada uma destas colunas. A seguir criam-se dois *arrays numpy*, um contendo os dados referentes às datas do set de treinamento e outro às de testes. Aplica-se o *PowerTransformer* aos dados de arrecadação e cria intervalos temporais considerando 5 janelas temporais por amostra, ou seja, contendo as arrecadações dos 5 dias anteriores ao dia em que a arrecadação será prevista.

```
@staticmethod
def cria_intervalos_temporais(np_array, n_intervalos=5):
    """ Dado um array NumPy com os valores diários, gera sequências
temporais com 3 dimensões para alimentarem a rede neural LSTM. """

    np_valores = np_array
    np_sequencia = np.empty((0, n_intervalos, 1))

    for i in range(n_intervalos, len(np_valores)):
        # Adiciona os itens que comporão uma sequência
        # Cada item é composto por uma sequência, n_intervalos intervalos
de tempo e 1 feature)
        np_item = np.empty((0, n_intervalos, 1))
        np_item = np.append(np_item, np_valores[(i-n_intervalos):i,
0].reshape(1, n_intervalos, 1), axis=0)
        # Adiciona uma sequência à lista de sequências
        np_sequencia = np.append(np_sequencia, np_item, axis=0)

    return np_sequencia
```

Figura 32 – Trecho do código em que são criados intervalos temporais da arrecadação

Foram adicionados 3 *callbacks* para serem executados durante o treinamento: *ModelCheckpoint* para salvar os pesos que resultarem em menor erro no *set* de testes; *EarlyStopping* para parar o treinamento quando não houver determinada redução do erro durante 50 *epochs* e *TensorBoard* para permitir o monitoramento do progresso durante o treinamento.

```
# Treina a rede neural LSTM com única variável quantitativa utilizando o
Power Transformer como scaler, já que foi o de melhor desempenho
for tributo in pd_arrecad_diaria['Tributo'].unique():
    # Utiliza método que extrai o dataset de teste idêntico ao utilizado no
```

```
Prophet
    df_treino, df_teste =
LSTMUtil.gera_teste_identico_prophet(arrecad_diaria[tributo],
pd_datas_testes.loc[tributo+' - Prophet - Univariável - Sem Remoção de
Outliers', 'Inicio'], pd_datas_testes.loc[tributo+' - Prophet - Univariável
- Sem Remoção de Outliers', 'Fim'])

    print('Tributo ' + tributo + ' - Início DF teste : ' + str(
        df_teste.reset_index().loc[0, 'Data']) + ' Fim DF teste : ' + str(
        df_teste.reset_index().loc[len(df_teste) - 1, 'Data']))
    df_treino = LSTMUtil.transforma_dataframe(df_treino, 'Data')
    df_teste = LSTMUtil.transforma_dataframe(df_teste, 'Data')

    # Faz o Label Encoder do dia e mês (apesar de dia e mês serem
numéricos, o Label Encoder inicia a contagem em 0 ao invés de 1)
    encoder_dia = LabelEncoder()
    dia_treino_enc = encoder_dia.fit_transform(df_treino['Dia'].values)
    dia_teste_enc = encoder_dia.transform(df_teste['Dia'].values)
    encoder_mes = LabelEncoder()
    mes_treino_enc = encoder_mes.fit_transform(df_treino['Mes'].values)
    mes_teste_enc = encoder_mes.transform(df_teste['Mes'].values)
    encoder_dia_semana = LabelEncoder()
    dia_semana_treino_enc =
encoder_dia_semana.fit_transform(df_treino['Dia_Semana'].values)
    dia_semana_teste_enc =
encoder_dia_semana.transform(df_teste['Dia_Semana'].values)

    # Concatena os valores para servirem de input para o modelo
    np_dia_mes_treino = np.concatenate((dia_treino_enc.reshape(-1, 1),
mes_treino_enc.reshape(-1, 1), dia_semana_treino_enc.reshape(-1, 1)),
axis=1)[5:]
    np_dia_mes_teste = np.concatenate((dia_teste_enc.reshape(-1, 1),
mes_teste_enc.reshape(-1, 1), dia_semana_teste_enc.reshape(-1, 1)),
axis=1)[5:]

    # Power Transformer (yeo-johnson)
    pwr_scaler = PowerTransformer()
    valor_treino_pwr =
pwr_scaler.fit_transform(df_treino['Valor'].values.reshape(-1, 1))
    valor_teste_pwr =
pwr_scaler.transform(df_teste['Valor'].values.reshape(-1, 1))

    saida_treino = valor_treino_pwr[5:]
    saida_teste = valor_teste_pwr[5:]

    # Cria intervalos temporais com 3 dimensões para servirem de input à
rede LSTM
    valor_arrecadacao_serie_temporal_lstm_treino =
LSTMUtil.cria_intervalos_temporais(valor_treino_pwr)
    valor_arrecadacao_serie_temporal_lstm_teste =
LSTMUtil.cria_intervalos_temporais(valor_teste_pwr)

    checkpoint =
ModelCheckpoint('checkpoint_regressor_'+tributo+'_univariado.hdf5',
monitor='loss', verbose=2,
                                save_best_only=True,
save_weights_only=False,
                                mode='auto', period=1)

    early_stopping = EarlyStopping(monitor='loss', min_delta=0.001,
patience=50)

    %load_ext tensorboard
    %tensorboard --logdir logs/scalars
```

```
    logdir = "logs/scalars/"
    tensorboard_callback = TensorBoard(log_dir=logdir, profile_batch =
100000000)

    model = LSTMUnivariada(df_treino)
    model.compile(optimizer=ko.Adam(lr=0.1), loss='mse')
    model.fit([np_dia_mes_treino,
valor_arrecadacao_serie_temporal_lstm_treino], saida_treino,
validation_data=([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste], saida_teste),
              epochs=1000, batch_size=50, callbacks=[checkpoint,
tensorboard_callback, early_stopping])
    print(model.summary())

    # Carrega o melhor modelo salvo pelo Checkpoint
    model.load_weights('checkpoint_regressor_'+tributo+'_univariado.hdf5')

    # Avalia a predição do test set
    pwr_pred = model.predict([np_dia_mes_teste,
valor_arrecadacao_serie_temporal_lstm_teste])
    rmse = mean_squared_error(pwr_scaler.inverse_transform(saida_teste),
pwr_scaler.inverse_transform(pwr_pred)) ** (1 / 2)
    mae = mean_absolute_error(pwr_scaler.inverse_transform(saida_teste),
pwr_scaler.inverse_transform(pwr_pred))

    # Preenche dataframe com os valores para comparação com os resultados
do Prophet
    pd_performance.loc[tributo+' - LSTM - Univariável', 'MAE'] = mae
    pd_performance.loc[tributo+' - LSTM - Univariável', 'RMSE'] = rmse
    pd_datas_testes.loc[tributo+' - LSTM - Univariável', 'Inicio'] =
df_teste.reset_index().loc[5, 'Data']
    pd_datas_testes.loc[tributo+' - LSTM - Univariável', 'Fim'] =
df_teste.reset_index().loc[len(df_teste)-1, 'Data']
    pd_datas_treinos.loc[tributo+' - LSTM - Univariável', 'Inicio'] =
df_treino.reset_index().loc[5, 'Data']
    pd_datas_treinos.loc[tributo+' - LSTM - Univariável', 'Fim'] =
df_treino.reset_index().loc[len(df_treino)-1, 'Data']

    # Plota as predições em comparação com os valores reais
    fig, (sub1) = plt.subplots(1, 1, sharex=True)
    pred, = plt.plot(df_teste[5:]['Data'],
pwr_scaler.inverse_transform(pwr_pred), c='blue', label='Predito')
    real = plt.scatter(df_teste[5:]['Data'], df_teste[5:]['Valor'], s=3,
c='orange')
    plt.legend([pred, real],
               ['Predito', 'Real'],
               fontsize=8)
    fig.autofmt_xdate()
    plt.xlabel('Data')
    plt.ylabel('Valor (R$)')
    plt.title(tributo)
    plt.show()
```

Figura 33 – Trecho do código em que é treinada a rede neural LSTM

5.2.2.3. Estrutura da Rede Neural

Inicialmente serão utilizados os mesmos dados utilizados na predição pelo Prophet, ou seja, a arrecadação em determinado dia e a data a que ela se refere. Dado que as redes neurais LSTM possuem a capacidade de analisar os dados passados, serão utilizados também os dados das 5 últimas arrecadações, gerando assim um

dataframe de 3 dimensões, contendo a quantidade de amostras, a quantidade de janelas temporais (neste caso, 5) e a quantidade de *features* (neste caso, 1).

Já para as datas, elas serão segregadas em dia, mês e dia da semana e serão inseridas na rede neural, ao contrário das séries temporais de arrecadação, como uma entrada de duas dimensões. As datas poderiam até ser incorporadas às arrecadações, entretanto como datas são sequências bem definidas (por exemplo, sempre após o dia 25 será dia 26 e sempre depois de junho será julho), sua incorporação às sequências de arrecadação tornaria o aprendizado mais lento sem aumentar o aprendizado. Sendo assim a rede será alimentada com *dataframes* de duas e três dimensões.

	Dia_Semana	Dia	Mes	Ano	Data	Valor
2472	4	19	10	2018	2018-10-19	2.550776e+06
2473	0	22	10	2018	2018-10-22	4.839436e+06
2474	1	23	10	2018	2018-10-23	2.877066e+06
2475	2	24	10	2018	2018-10-24	2.229760e+06
2476	3	25	10	2018	2018-10-25	2.364328e+06
	...	...	...	...	...	...
3092	3	1	4	2021	2021-04-01	1.896179e+06
3093	0	5	4	2021	2021-04-05	2.368007e+06
3094	1	6	4	2021	2021-04-06	2.492219e+06
3095	2	7	4	2021	2021-04-07	2.101995e+06
3096	3	8	4	2021	2021-04-08	3.600000e+02

Figura 34 – Exemplos de entrada da rede neural LSTM, antes da separação em duas entradas de dimensões distintas

```
array([[25,  9,  4],
       [28,  9,  0],
       [29,  9,  1],
       ...,
       [ 5,  3,  1],
       [ 6,  3,  2],
       [ 7,  3,  3]], dtype=int64)
```

Figura 35 – Exemplos de entrada das datas na rede neural LSTM, contendo dia, mês e dia da semana, respectivamente

```
array([[[-1.25773406],
        [-1.09787706],
        [-0.96184849],
        [-1.04362998],
        [-0.27722393]],

       [[-1.09787706],
        [-0.96184849],
        [-1.04362998],
        [-0.27722393],
        [-1.08898924]],

       [[-0.96184849],
        [-1.04362998],
        [-0.27722393],
        [-1.08898924],
        [-0.79483831]],

       ...,
```

```
       [[ 1.56565373],
        [ 0.62283076],
        [ 1.02276447],
        [ 0.86245284],
        [ 0.86955776]],

       [[ 0.62283076],
        [ 1.02276447],
        [ 0.86245284],
        [ 0.86955776],
        [ 0.42315848]],

       [[ 1.02276447],
        [ 0.86245284],
        [ 0.86955776],
        [ 0.42315848],
        [ 0.57709887]]])
```

Figura 36 – Exemplos de entrada das arrecadações na rede neural LSTM com 3 dimensões, com os valores já padronizados pelo PowerTransformer

Para a criação da rede neural utilizar-se-á a biblioteca *Keras* com o *TensorFlow 2* como *backend*.

5.2.2.3.1. *One Hot Encoder* vs *Embedding*

Apesar de serem números, datas são variáveis categóricas, devendo ser codificadas adequadamente para que as redes neurais possam processá-las de maneira correta. Caso fosse inserido um determinado dia sem a devida codificação, a rede neural entenderia o dia 10, por exemplo, como superior em termos de arrecadação ao dia 3, o que pode não ser uma verdade. Para tanto, tem-se utilizado majoritariamente duas formas de codificação: *One Hot Encoder* (OHE) e *Embedding*.

O OHE gera uma coluna para cada categoria de dado, preenchendo a coluna correspondente à amostra com 1 e as demais com 0 (tornando-se *dummy variables*). Desta maneira teríamos 31 colunas para os dias, 12 para os meses e 7 para os dias da semana, transformando a coluna correspondente à data em 50 colunas, sendo que em apenas 3 delas por amostras os pesos da rede neural seriam efetivamente aplicados, uma vez que as demais possuiriam valor 0 e não contribuiriam na redução do erro do modelo.

Já a camada *Embedding* é aplicada diretamente à estrutura da rede neural, necessitando apenas que as variáveis sejam convertidas em números utilizando o método *LabelEncoder*. Durante o treinamento, a rede neural se encarregará de relacioná-las com as demais *features* do modelo criando relacionamentos multidimensionais para associá-las, transformando-as em vetores, permitindo associar, por

exemplo, que a arrecadação no dia 22 é semelhante à arrecadação no dia 29. Pelo seu dinamismo, optou-se pelo uso deste método no trabalho.

5.2.2.3.2. *Sequential API* vs *Subclassing*

O uso mais comum do *Keras* é utilizando a *Sequential API*, pois permite que os modelos sejam criados de maneira mais simples, bastando inserir as camadas em sequência e, por último, compilar o modelo. A *Sequential API* atende a grande maioria dos modelos de rede neural, entretanto ela não possui a capacidade de que sejam inseridos *dataframes* com dimensões distintas, que é o caso deste estudo, como descrito ao final do item 5.2.2.2.

Utilizar modelos adotando *subclassing* é, por sua vez, bem mais flexível que a *Sequential API*, permitindo que sejam inseridos *dataframes* de múltiplas dimensões como entrada da rede neural. Por outro lado é uma maneira bem mais complexa de criar modelos, necessitando da criação de uma classe específica que herda a classe *tf.keras.Model*. Nesta classe é criado o método *call*, responsável por "montar" a rede neural, definindo as interações entre cada camada. Neste trabalho, a rede neural com apenas uma variável quantitativa (arrecadação) é criada pela classe *LSTMUnivariada* do arquivo *ModelosNN.py*.

```
class LSTMUnivariada(tf.keras.Model):

    def __init__(self, df):
        super(LSTMUnivariada, self).__init__()
        self.df = df
        self.cria_rede_neural_univariada(df)
        self.valor = Dense(1, activation='linear', name='Valor')

        ''' Faz o "build" das camadas que compõem a rede neural. '''
        self.embedding_dia.build([None, 1])
        self.embedding_mes.build([None, 1])
        self.embedding_dia_semana.build([None, 1])
        self.dense_dia_mes_valor.build([None, 1])

        ''' O primeiro item da lista se refere ao mês, dia e dia da semana,
já o segundo item se refere à LSTM, com 5 períodos (5 dias anteriores à
predição) e uma variável (valor). '''
        self.build([(None, 3), (None, 5, 1)])

    def cria_rede_neural_univariada(self, df):
        """ Cria rede neural "univariada" usando o Keras Subclass,
retornando um modelo
        do Keras. """

        dias_distintos = df['Dia'].unique()
        meses_distintos = df['Mes'].unique()
        dias_semana_distintos = df['Dia_Semana'].unique()

        ''' Adiciona as camadas ao modelo. '''
```

```
        self.embedding_dia = Embedding(name='dia_embedding',
input_length=1,
                                       input_dim=len(dias_distintos),

output_dim=int(round(len(dias_distintos) ** 0.25, 0)))
        self.flatten_dia = Flatten()
        self.embedding_mes = Embedding(name='mes_embedding',
input_length=1,
                                       input_dim=len(meses_distintos),

output_dim=int(round(len(meses_distintos) ** 0.25, 0)))
        self.flatten_mes = Flatten()
        self.embedding_dia_semana = Embedding(name='dia_semana_embedding',
input_length=1,

input_dim=len(dias_semana_distintos),

output_dim=int(round(len(dias_semana_distintos) ** 0.25, 0)))
        self.flatten_dia_semana = Flatten()
        self.concatenate_dia_mes = Concatenate(axis=-1,
name='dia_mes_concatenate')
        self.dense_dia_mes = Dense(2, activation='relu',
name='dia_mes_dense')
        self.lstm_valor = LSTM(1, name='valor_lstm')
        self.dense_valor = Dense(1, activation='relu', name='valor_dense')
        self.concatenate_dia_mes_valor = Concatenate(axis=-1,
name='dia_mes_valor_concatenate')
        self.dense_dia_mes_valor = Dense(1, activation='sigmoid',
name='dia_mes_valor_dense')

    def call(self, inputs, **kwargs):
        # inputs[0] são os dados de dia e mês
        dia_mes_tensor = tf.convert_to_tensor(inputs[0])

        # inputs[1] são os dados do valor arrecadado
        valor_tensor = tf.convert_to_tensor(inputs[1])

        # dia_mes_tensor[:, 0] são os dados do dia
        # dia_mes_tensor[:, 1] são os dados do mês
        # dia_mes_tensor[:, 2] são os dados do dia da semana
        flt_dia = self.flatten_dia(self.embedding_dia(dia_mes_tensor[:,
0]))
        flt_mes = self.flatten_mes(self.embedding_dia(dia_mes_tensor[:,
1]))
        flt_dia_semana =
self.flatten_dia_semana(self.embedding_dia(dia_mes_tensor[:, 2]))
        concat_dia_mes = self.concatenate_dia_mes([flt_dia, flt_mes,
flt_dia_semana])
        dense_dia_mes = self.dense_dia_mes(concat_dia_mes)
        lstm_valor = self.lstm_valor(valor_tensor)
        dense_valor = self.dense_valor(lstm_valor)
        dia_mes_valor = self.concatenate_dia_mes_valor([dense_dia_mes,
dense_valor])

        return self.valor(dia_mes_valor)

    def get_config(self):
        pass
```

Figura 37 – Trecho do código em que é criada a rede neural LSTM com uma variável quantitativa

São adicionadas 3 camadas *Embedding* para dia, mês e dia da semana com duas dimensões cada. A quantidade de dimensões é definida pela raiz quarta do

número de categorias existentes[9]. Caso o resultado não seja inteiro, o valor é arredondado para o inteiro mais próximo.

$$numero\_dimensoes = \sqrt[4]{numero\_categorias}$$

Desta maneira temos 31 categorias para dias, 12 para meses e 7 para dias da semana que, multiplicadas pelas 2 dimensões obtidas pelo cálculo acima, obtemos a quantidade de parâmetros exibidas no resumo do modelo abaixo. Após o "achatamento" de cada uma dessas camadas *Embedding* através da camada *Flatten*, reduzindo o número de dimensões, os tensores representativos de dia, mês e dia da semana são concatenados e finalmente utiliza a camada *Dense* para conectá-los. Foi utilizado o ativador *Rectified Linear Unit* (ReLU) por apresentar melhor performance durante o *backpropagation*[10]. O resultado da conexão será concatenado mais à frente com a saída da camada LSTM da rede neural.

Com relação à camada LSTM, esta receberá como entrada *samples* com 1 *feature* (arrecadação) e 5 janelas temporais (arrecadação nos 5 dias anteriores). A saída da camada LSTM será, por sua vez, conectada a uma camada *Dense*. A saída desta camada *Dense* será concatenada à saída da camada *Dense* aplicada às datas e finalmente o resultado da concatenação será conectado à última camada *Dense* com apenas uma saída, que será a arrecadação predita nas dimensões do PowerTransformer (para exibi-la em termos absolutos deve-se usar o método *inverse_transform* do *PowerTransformer*). Seguem abaixo o resumo da rede neural obtido pelo método *summary* do *tf.keras.Model*[11] e o gráfico representativo da arquitetura obtido *pelo TensorBoard*.

```
Model: "lstm_univariada_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dia_embedding (Embedding)    multiple                  62
_________________________________________________________________
flatten_12 (Flatten)         multiple                  0
_________________________________________________________________
mes_embedding (Embedding)    multiple                  24
_________________________________________________________________
flatten_13 (Flatten)         multiple                  0
```

[9] Disponível em: <https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html>. Acesso em 29 de abril de 2021.

[10] (Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning*. MIT Press, 2017, p. 226)

[11] A coluna *Output Shape* está exibindo apenas *multiple* devido a um *bug* do Keras ao utilizar modelos criados por herança (*subclassing*) da classe tf.keras.Model. O relato do problema pode ser consultado em: <https://github.com/tensorflow/tensorflow/issues/25036>. Acesso em 29 de abril de 2021.

```
_________________________________________________________________
dia_semana_embedding (Embedd multiple                  14
_________________________________________________________________
flatten_14 (Flatten)         multiple                  0
_________________________________________________________________
dia_mes_concatenate (Concate multiple                  0
_________________________________________________________________
dia_mes_dense (Dense)        multiple                  14
_________________________________________________________________
valor_lstm (LSTM)            multiple                  12
_________________________________________________________________
valor_dense (Dense)          multiple                  2
_________________________________________________________________
dia_mes_valor_concatenate (C multiple                  0
_________________________________________________________________
dia_mes_valor_dense (Dense)  multiple                  2
_________________________________________________________________
Valor (Dense)                multiple                  4
=================================================================
Total params: 134
Trainable params: 134
Non-trainable params: 0
_________________________________________________________________
```

Figura 38 – Estrutura da rede neural obtida através do método summary da classe tf.keras.Model

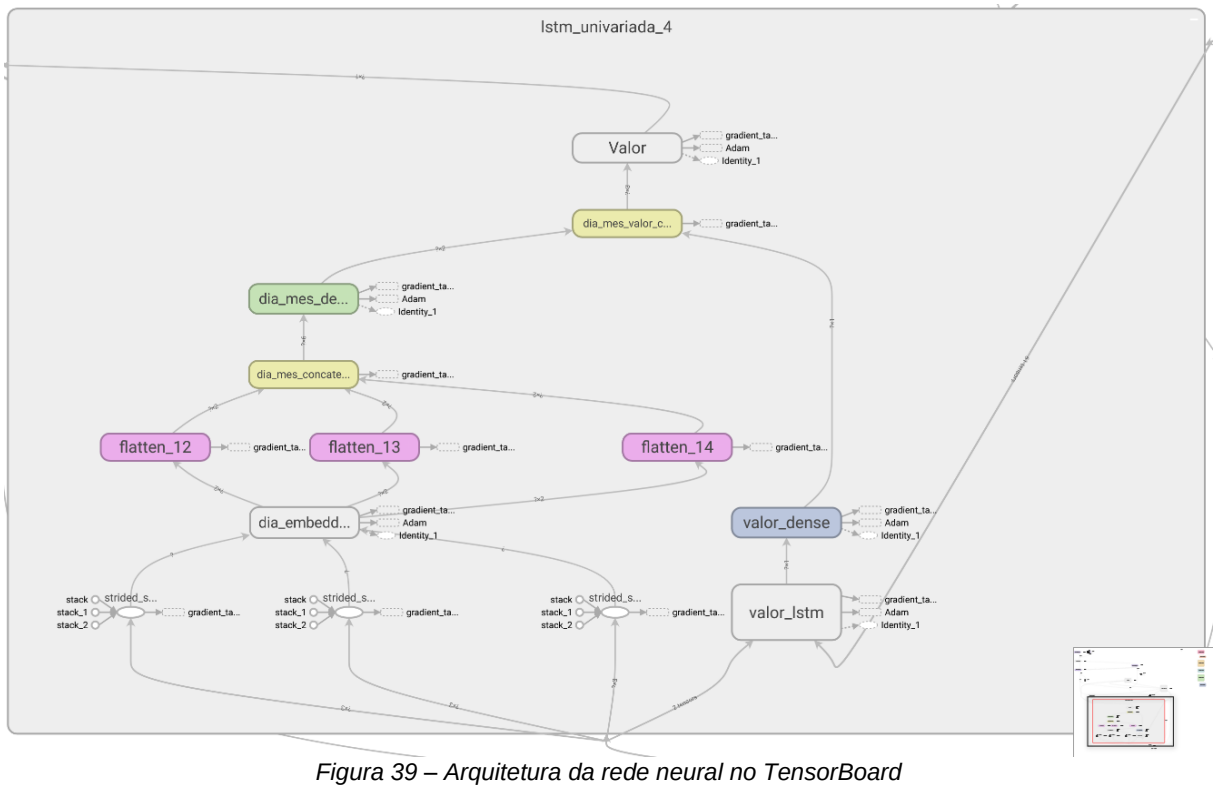

Figura 39 – Arquitetura da rede neural no TensorBoard

5.2.2.4. Treinamento da rede neural

O treinamento utilizará a tecnologia CUDA® das placas gráficas da NVIDIA, acionada automaticamente pelo *TensorFlow*, com objetivo de acelerar os cálculos dos tensores. Após o treinamento, os modelos executarão a predição do mesmo *set* de testes utilizado nos itens 5.2.1.1 e 5.2.2.2 referentes ao *Prophet*. O equipamento utilizado possui a seguinte configuração:

Processador: Intel Core i7 7700-HQ 2.80 GHz

Placa de Vídeo: NVIDIA GeForce GTX 1050 Ti 4 GB GDDR5, 768 núcleos CUDA®

Memória RAM: 16 GB

Armazenamento: SSD M.2 NVMe Corsair Force Serie 480 GB

O treinamento será realizado em, no máximo, 1000 *epochs* (devido ao *EarlyStopping* pode haver interrupção antes da execução de todas as *epochs*), sendo que em cada *epoch* serão utilizadas *batches* com 50 *samples*.

5.2.2.5. Resultados das predições

Realizados os treinamentos para cada tributo, serão realizadas as predições para os períodos de teste (idênticos aos utilizados no Prophet) e suas respectivas plotagens.

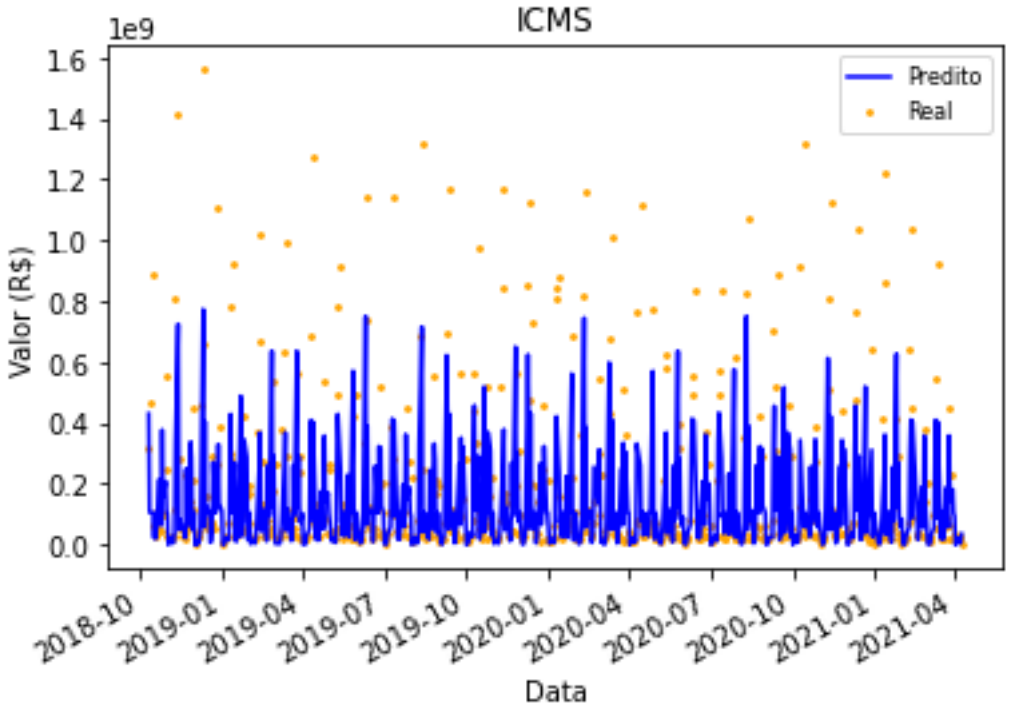

Figura 40 – Predição da arrecadação do ICMS

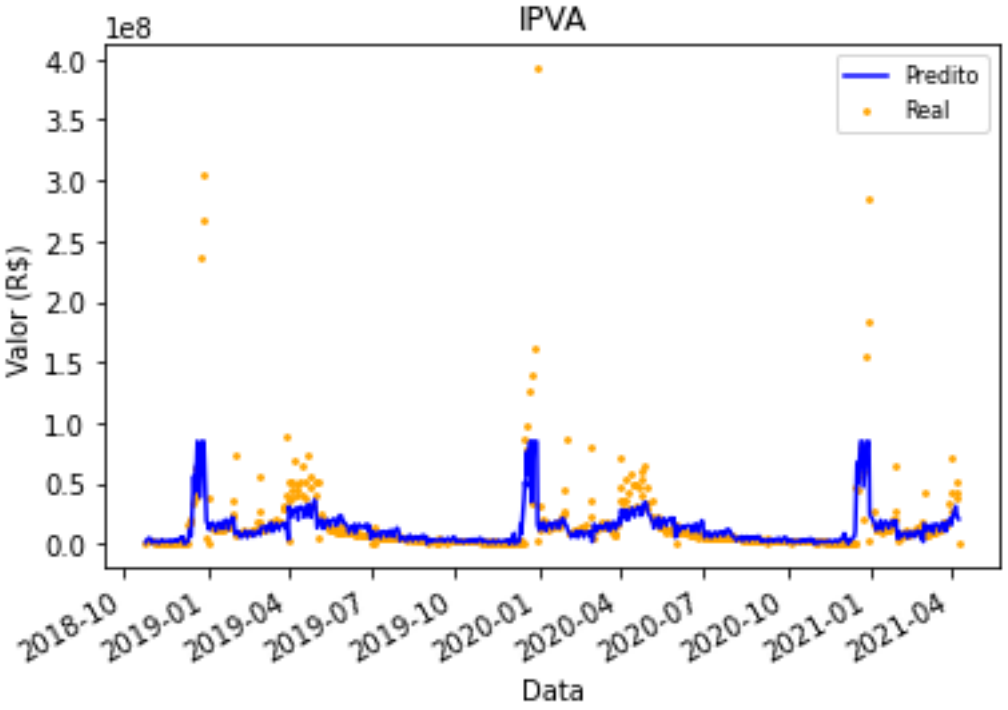

Figura 41 – Predição da arrecadação do IPVA

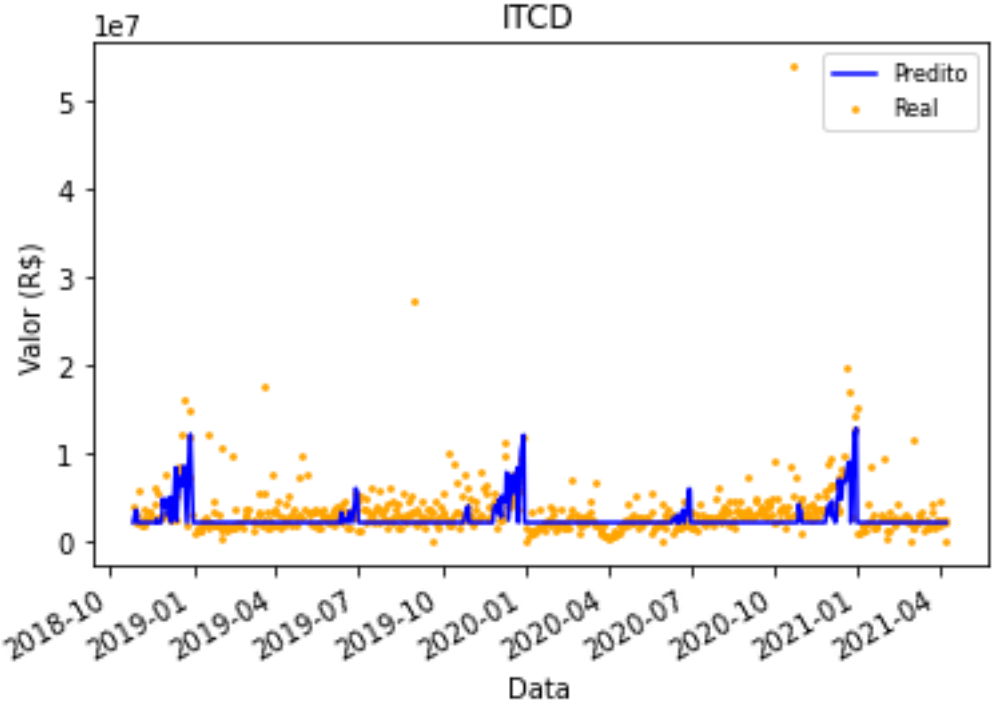

Figura 42 – Predição da arrecadação do ITCD

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	207.152.563,71	277.909.138,33	0,74539673
IPVA	23.344.215,09	33.042.123,48	0,70649863
ITCD	3.143.160,58	3.311.754,46	0,94909228

Figura 43 – Root Mean Squared Error dos resíduos das predições da rede neural LSTM

Tributo	Data início testes	Data fim testes
ICMS	10/10/2018	08/04/2021
IPVA	25/10/2018	08/04/2021
ITCD	26/10/2018	08/04/2021

Figura 44 – Datas correspondentes ao intervalo de testes dos modelos do Prophet e LSTM

Evidencia-se que, da mesma forma que as predições realizadas pelo Prophet, o RMSE das predições pela rede neural LSTM é diretamente proporcional ao desvio padrão das séries temporais, sendo que o ICMS continua apresentando maior RMSE devido ao seu alto desvio padrão.

Tributo	RMSE/Desvio padrão – Prophet (A)	RMSE/Desvio padrão – LSTM (B)	1-A/B
ICMS	0,986897	0,74539673	0,244706661
IPVA	0,787135	0,70649863	0,102442872
ITCD	0,958725	0,94909228	0,010047428

Figura 45 – Comparativo do desempenho entre o Prophet – sem remoção de outliers – e a rede neural LSTM

No entanto, ao comparar a razão entre o RMSE e o desvio padrão das séries temporais, percebe que a rede neural LSTM foi superior ao Prophet em todos os tributos, com destaque para o ICMS, em que a LSTM teve performance aproximadamente 24,5% superior. É possível perceber que o Prophet se comporta melhor em séries temporais com maior sazonalidade (vide IPVA), pois no seu funcionamento padrão, sem adicionar outras variáveis preditoras, ele trabalha apenas com tendência e sazonalidade. Já a rede neural LSTM foi montada para trabalhar com sazonalidade e, ainda, receber dados das arrecadações anteriores dos tributos, permitindo ao modelo que identifique tendências recentes de arrecadação que não podem ser identificadas apenas com a data informada. Redes neurais artificiais, principalmente as redes neurais recorrentes como a LSTM, são difíceis (para não dizer impossíveis) de interpretar, entretanto, no caso do ICMS, a elevada redução de erro em relação ao Prophet se deve provavelmente à baixa importância que as datas têm para predição da arrecadação (uma vez que não há baixa sazonalidade para este tributo), dando maior importância aos dados de arrecadação, que, neste trabalho, utilizando o funcionamento

padrão do Prophet, foram tratados apenas pela rede LSTM. Para o ITCD o ganho da LSTM em relação ao Prophet foi irrelevante, não justificando o uso de um modelo complexo e com alto custo de processamento como a LSTM.

5.3. Modelos com múltiplas variáveis quantitativas

Analisados os modelos que utilizam apenas as datas e a respectiva arrecadação para prever a arrecadação futura, parte-se para uma análise com outras variáveis preditivas para verificação de como cada modelo se comporta frente a elas. Neste tópico será analisado apenas o ICMS pois, além de ser o tributo de maior importância para os estados, é aquele que possui maior número de informações públicas que podem ser utilizadas como variáveis preditoras. Neste caso seria inútil utilizar as mesmas variáveis preditoras para todos os tributos, uma vez que cada um deles possui fatos geradores diferentes entre si. O ICMS tem, por exemplo, seu fato gerador vinculado à circulação de mercadorias e serviços com intuito comercial, sendo assim é um tributo que está intrinsecamente correlacionado à atividade econômica como um todo. Já o IPVA está relacionado à propriedade de veículos automotores no primeiro dia de cada ano e a emplacamentos de veículos novos. O ITCD, por sua vez, relaciona-se com a conclusão de processos judiciais de inventários, gerando a partilha. A restrição deste tópico ao ICMS não significa que o mesmo não poderia ser feito com os outros dois tributos, entretanto necessitaria de informações que não são publicadas pelo estado do Rio Grande do Sul com periodicidade compatível à periodicidade diária da arrecadação.

5.3.1. As novas variáveis preditoras

Na tentativa de reduzir o erro nas predições da arrecadação do ICMS, serão utilizadas mais três variáveis preditoras: PIB trimestral do estado do Rio Grande do Sul no trimestre anterior (obtido no sítio do Departamento de Economia e Estatística do RS), PIB mensal brasileiro no mês anterior (obtido no sítio do IPEA Data, tendo como fonte o Banco Central do Brasil), quantidade de admissões no mês anterior e quantidade de demissões no mês anterior (obtidos no sítio do IPEA Data, tendo como fonte o Caged). No caso do PIB Mensal brasileiro, os valores são estimados pelo Bacen com base no PIB trimestral, não sendo, portanto, valores absolutos. Há que se dizer também que existem variáveis preditoras não-públicas de melhor qualidade, como a emissão de notas fiscais em operações de venda e revenda no mês anterior (que evidenciam a ocorrência do fato gerador) e índices de inadimplência de

empresas no mês anterior (que permitem predizer se os tributos serão efetivamente arrecadados pelo estado) em órgãos de proteção ao crédito, como SPC e Serasa. Os dados serão obtidos pelos métodos *download_pib_br*, *download_dados_emprego* e *download_pib_rs* da classe *DownloadDados*.

Com relação à periodicidade dessas novas variáveis preditoras, não foi possível encontrar nenhuma delas em intervalos diários, o que é até compreensível, dado que são valores de medição bastante complexa. Para permitir que os dados diários de arrecadação possuam relacionamento um para um com as novas variáveis preditoras, eles serão relacionados com o PIB mensal brasileiro, admissões e demissões do mês imediatamente anterior e com o PIB mensal do RS no trimestre imediatamente anterior. Os relacionamentos serão realizados pelos métodos *adiciona_pib_br*, *adiciona_dados_emprego* e *adiciona_pib_rs* da classe *ProphetUtil* do arquivo *ModelosUtil.py*.

5.3.1.1. Prophet

Para adicionar mais variáveis preditoras aos modelos do Prophet, o procedimento é simples, bastando invocar o método *add_regressor* do objeto representativo do modelo e indicar a coluna com os valores da variável.

```
pd_prophet_icms =
ProphetUtil.transforma_dataframe(pd_arrecad_diaria[pd_arrecad_diaria['Tribu
to']=='ICMS'], ['Data', 'Valor'])

# Baixa os dados do PIB trimestral do Rio Grande do Sul (em valores
atualizados)
pd_pib_rs_trimestral = DownloadDados.download_pib_rs()
pd_pib_br_mensal = DownloadDados.download_pib_br()
pd_pib_br_mensal = CorrigeValores.corrige_inflacao_pib(pd_pib_br_mensal,
igp)
pd_emprego_adm_dem_mensal = DownloadDados.download_dados_emprego()

# Adiciona os dados do PIB do RS(trimestre anterior) ao dataframe
pd_prophet_icms = ProphetUtil.adiciona_pib_rs(pd_prophet_icms,
pd_pib_rs_trimestral)
# Adiciona os dados do PIB do Brasil(estimativa mês anterior) ao dataframe
pd_prophet_icms = ProphetUtil.adiciona_pib_br(pd_prophet_icms,
pd_pib_br_mensal)
# Adiciona os dados de emprego do Brasil(mês anterior) ao dataframe
pd_prophet_icms = ProphetUtil.adiciona_dados_emprego(pd_prophet_icms,
pd_emprego_adm_dem_mensal)

# Cria modelo com múltiplas variáveis quantitativas utilizando o Facebook
Prophet
prophet = Prophet(daily_seasonality=True)
pd_prophet = pd_prophet_icms
df_treino, df_teste = ProphetUtil.divide_treino_teste(pd_prophet)
df_teste.reset_index(drop=True, inplace=True)
prophet.add_regressor('ADMISSOES_MES_ANTERIOR')
```

```
prophet.add_regressor('DEMISSOES_MES_ANTERIOR')
prophet.add_regressor('PIB_BR_MES_ANTERIOR')
prophet.add_regressor('PIB_RS_TRIMESTRE_ANTERIOR')
prophet.fit(df_treino)
predito = prophet.predict(pd.DataFrame(df_teste[['ds',
'ADMISSOES_MES_ANTERIOR', 'DEMISSOES_MES_ANTERIOR', 'PIB_BR_MES_ANTERIOR',
'PIB_RS_TRIMESTRE_ANTERIOR']]))
```

Figura 46 – Trecho do código em que é criado modelo do Prophet com múltiplas variáveis quantitativas

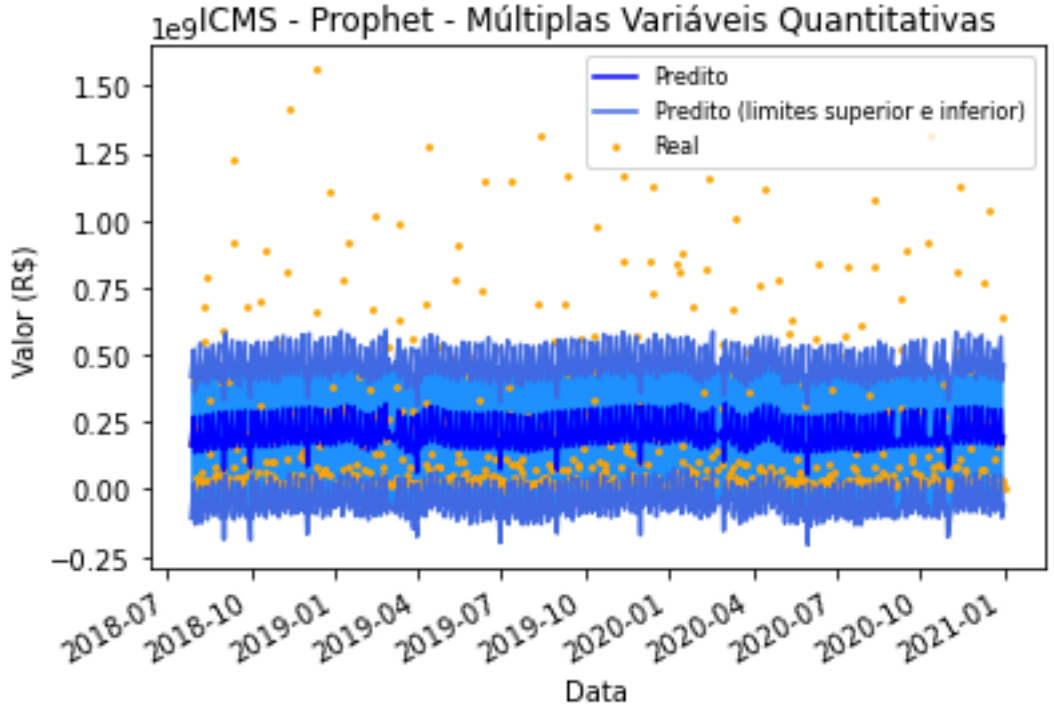

Figura 47 – Plotagem da predição do Prophet do ICMS utilizando múltiplas variáveis quantitativas preditoras

Tributo	Data início testes	Data fim testes
ICMS	27/07/2018	31/12/2020

Figura 48 – Datas do período de testes do modelo multivariado do Prophet

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	274.007.378,80	279.074.007,43	0,9818449

Figura 49 – Resultados do modelo multivariado no período de testes entre 27/07/2018 e 31/12/2020

Realizada a predição, foram gerados os dados no mesmo formato que os dos modelos univariados, entretanto ainda não podemos tirar conclusões se as novas variáveis preditoras foram capazes de melhorar o modelo preditivo (em comparação com os testes realizados no item 5.2.1.2), uma vez que, devido à indisponibilidade do PIB Trimestral do RS para os trimestres anteriores ao primeiro e segundo trimestre de 2021 não foi possível a montagem do *dataset* sem *missing data* no mesmo período

de testes do modelo com apenas uma variável quantitativa. Até poderia ser utilizada alguma técnica para preencher os valores ausentes, entretanto como o PIB possui variância significativa, qualquer erro poderia gerar grande discrepância entre o resultado predito o real. Sendo assim, para fins de comparação, preferiu-se gerar um novo modelo univariado seguindo as mesmas datas do modelo multivariado.

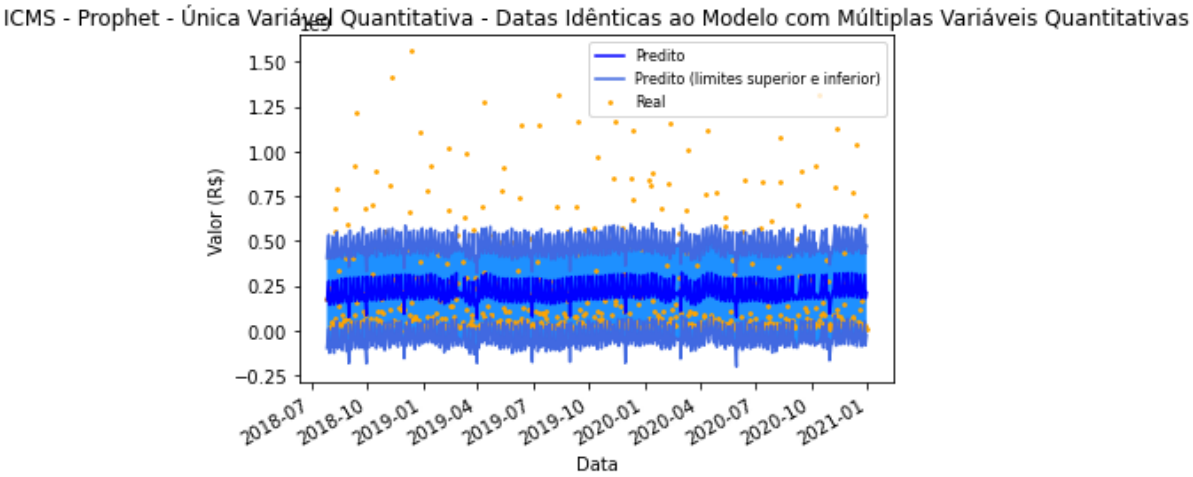

Figura 50 – Plotagem da predição do Prophet do ICMS utilizando única variável quantitativa, no período entre 27/07/2018 e 31/12/2020

Tributo	Data início testes	Data fim testes
ICMS	27/07/2018	31/12/2020

Figura 51 – Datas do período de testes do modelo univariado do Prophet

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	274.548.032,84	279.074.007,43	0,98378217

Figura 52 – Resultados do modelo univariado no período de testes entre 27/07/2018 e 31/12/2020

Utilizados os mesmos períodos de testes, percebe-se uma ligeira piora do modelo com múltiplas variáveis quantitativas em relação àquele que utiliza apenas as datas como variável preditora. Dado que o Prophet foi criado para tratar tendência e sazonalidade, espera-se que novas variáveis inseridas como regressoras no model tenham papel coadjuvante na redução dos resíduos.

5.3.1.2. LSTM

O modelo para a rede neural LSTM com múltiplas variáveis quantitativas será praticamente o mesmo utilizado pelo modelo com única variável e ilustrado na Figura 37. A diferença entre eles será a adição de novas colunas no *dataframe* de entrada

de duas dimensões da rede neural, colunas estas que representam as novas variáveis regressoras. Com relação ao tratamento das novas variáveis, houve uma pequena necessidade de preenchimento de *missing values*, como demonstrado no item 3. Para a normalização, assim como nos valores de arrecadação, foi utilizado o *PowerTransformer*. Os demais parâmetros de treinamento (*call-backs*, *batch size*, *epochs* etc.) foram os mesmos utilizados pelo modelo com única variável quantitativa.

```
class LSTMMultivariada(tf.keras.Model):

    def __init__(self, df):
        super(LSTMMultivariada, self).__init__()
        self.df = df
        self.cria_rede_neural_multivariada(df)
        self.valor = Dense(1, activation='linear', name='Valor')

        ''' O primeiro item da lista se refere ao mês, dia, dia da semana,
PIB do RS no trimestre anterior,
        PIB do Brasil do mês anterior, admissões no mês anterior e
demissões no mês anterior,
        já o segundo item se refere à LSTM, com 5 períodos (5 dias
anteriores à predição) e uma variável (valor). '''
        self.build([(None, 7), (None, 5, 1)])

    def cria_rede_neural_multivariada(self, df):
        """ Cria rede neural "multivariada" usando o Keras Subclass,
retornando um modelo
        do Keras. """

        dias_distintos = df['Dia'].unique()
        meses_distintos = df['Mes'].unique()
        dias_semana_distintos = df['Dia_Semana'].unique()

        self.embedding_dia = Embedding(name='dia_embedding',
input_length=1,
                                       input_dim=len(dias_distintos),

output_dim=int(round(len(dias_distintos) ** 0.25, 0)))
        self.flatten_dia = Flatten()
        self.embedding_mes = Embedding(name='mes_embedding',
input_length=1,
                                       input_dim=len(meses_distintos),

output_dim=int(round(len(meses_distintos) ** 0.25, 0)))
        self.flatten_mes = Flatten()
        self.embedding_dia_semana = Embedding(name='dia_semana_embedding',
input_length=1,

input_dim=len(dias_semana_distintos),

output_dim=int(round(len(dias_semana_distintos) ** 0.25, 0)))
        self.flatten_dia_semana = Flatten()
        self.flatten_pib_rs = Flatten()
        self.flatten_pib_br = Flatten()
        self.flatten_admissoes = Flatten()
        self.flatten_admissoes = Flatten()
        self.concatenate_dia_mes = Concatenate(axis=-1,
name='dia_mes_concatenate')
        self.dense_dia_mes = Dense(2, activation='relu',
```

```
name='dia_mes_dense')
        self.lstm_valor = LSTM(1, name='valor_lstm')
        self.dense_valor = Dense(1, activation='relu', name='valor_dense')
        self.concatenate_dia_mes_valor = Concatenate(axis=-1,
name='dia_mes_valor_concatenate')
        self.dense_dia_mes_valor = Dense(1, activation='sigmoid',
name='dia_mes_valor_dense')

    def call(self, inputs, **kwargs):
        # inputs[0] são os dados de dia e mês
        dia_mes_tensor = tf.convert_to_tensor(inputs[0])

        # inputs[1] são os dados do valor arrecadado
        valor_tensor = tf.convert_to_tensor(inputs[1])

        # dia_mes_tensor[:, 0] são os dados do dia
        # dia_mes_tensor[:, 1] são os dados do mês
        # dia_mes_tensor[:, 2] são os dados do dia da semana
        # dia_mes_tensor[:, 3] são os dados do PIB trimestral do RS no
trimestre anterior
        # dia_mes_tensor[:, 4] são os dados do PIB mensal do Brasil no mês
anterior
        # dia_mes_tensor[:, 5] são os dados de admissões no mês anterior
        # dia_mes_tensor[:, 6] são os dados de demissões no mês anterior
        flt_dia = self.flatten_dia(self.embedding_dia(dia_mes_tensor[:,
0]))
        flt_mes = self.flatten_mes(self.embedding_dia(dia_mes_tensor[:,
1]))
        flt_dia_semana =
self.flatten_dia_semana(self.embedding_dia(dia_mes_tensor[:, 2]))
        flt_pib_rs = self.flatten_pib_rs(dia_mes_tensor[:, 3])
        flt_pib_br =  self.flatten_pib_br(dia_mes_tensor[:, 4])
        flt_admissoes =  self.flatten_admissoes(dia_mes_tensor[:, 5])
        flt_demissoes = self.flatten_admissoes(dia_mes_tensor[:, 6])

        concat_dia_mes = self.concatenate_dia_mes([flt_dia, flt_mes,
flt_dia_semana, flt_pib_rs, flt_pib_br, flt_admissoes, flt_demissoes])
        dense_dia_mes = self.dense_dia_mes(concat_dia_mes)
        lstm_valor = self.lstm_valor(valor_tensor)
        dense_valor = self.dense_valor(lstm_valor)
        dia_mes_valor = self.concatenate_dia_mes_valor([dense_dia_mes,
dense_valor])

        return self.valor(dia_mes_valor)

    def get_config(self):
        pass
```

Figura 53– Trecho do código em que é criada a rede neural LSTM com múltiplas variáveis quantitativas

```
Model: "lstm_multivariada_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dia_embedding (Embedding)    multiple                  62
_________________________________________________________________
flatten_34 (Flatten)         multiple                  0
_________________________________________________________________
mes_embedding (Embedding)    multiple                  24
_________________________________________________________________
flatten_35 (Flatten)         multiple                  0
_________________________________________________________________
dia_semana_embedding (Embedd multiple                  14
_________________________________________________________________
flatten_36 (Flatten)         multiple                  0
```

```
_________________________________________________________________
flatten_37 (Flatten)         multiple                  0
_________________________________________________________________
flatten_38 (Flatten)         multiple                  0
_________________________________________________________________
flatten_40 (Flatten)         multiple                  0
_________________________________________________________________
dia_mes_concatenate (Concate multiple                  0
_________________________________________________________________
dia_mes_dense (Dense)        multiple                  22
_________________________________________________________________
valor_lstm (LSTM)            multiple                  12
_________________________________________________________________
valor_dense (Dense)          multiple                  2
_________________________________________________________________
dia_mes_valor_concatenate (C multiple                  0
_________________________________________________________________
dia_mes_valor_dense (Dense)  multiple                  2
_________________________________________________________________
Valor (Dense)                multiple                  4
=================================================================
Total params: 142
Trainable params: 142
Non-trainable params: 0
```

Figura 54 – Estrutura da rede neural obtida através do método summary da classe tf.keras.Model

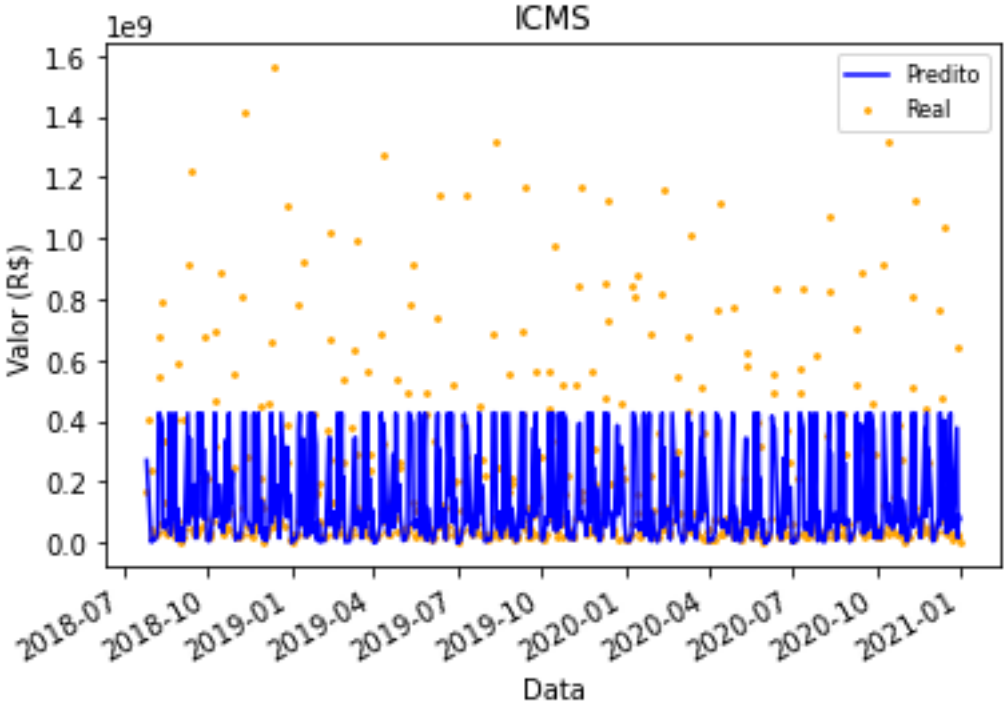

Figura 55 – Plotagem da predição LSTM do ICMS utilizando múltiplas variáveis quantitativas preditoras

Tributo	Data início testes	Data fim testes
ICMS	27/07/2018	31/12/2020

Figura 56 – Datas do período de testes do modelo multivariado LSTM

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	201.564.796,27	279.074.007,43	0,72226288

Figura 57 – Resultados do modelo multivariado no período de testes entre 27/07/2018 e 31/12/2020

Da mesma forma que o modelo do Prophet, não é possível, ainda, realizar a comparação com o modelo com apenas uma variável quantitativa (item 5.2.2.5), uma vez que os períodos de testes são distintos. Para tanto, faz-se necessário treinamento e predição do modelo univariado em períodos em que não haja *missing values* de nenhuma das variáveis preditoras, permitindo a comparação entre modelos distintos no mesmo período.

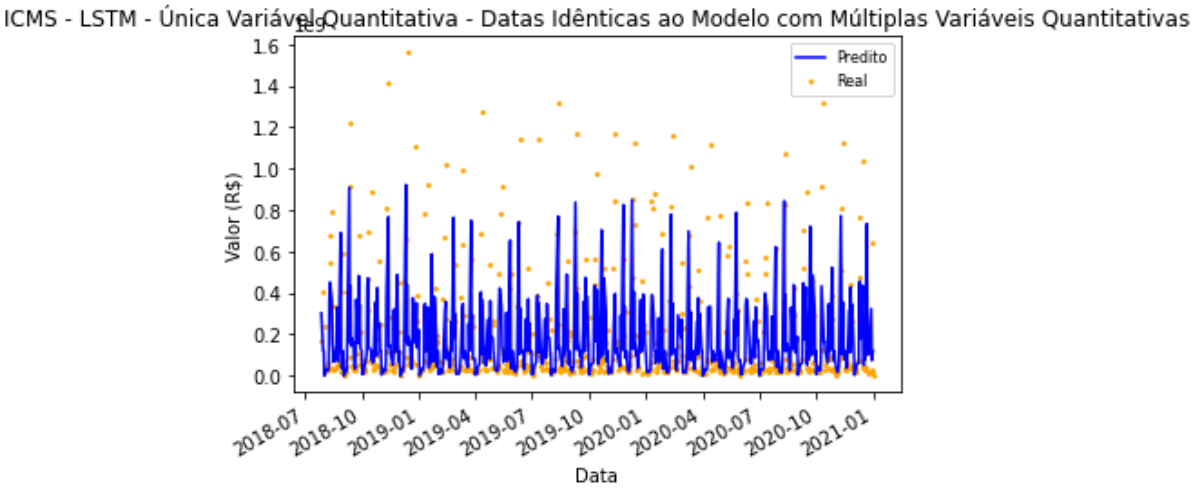

Figura 58 – Plotagem da predição do Prophet do ICMS utilizando única variável quantitativa, no período entre 27/07/2018 e 31/12/2020

Tributo	Data início testes	Data fim testes
ICMS	27/07/2018	31/12/2020

Figura 59 – Datas do período de testes do modelo univariado LSTM

Tributo	RMSE (A)	Desvio Padrão set de testes (B)	A/B = (C)
ICMS	207.916.464,69	279.074.007,43	0,745022679

Figura 60 – Resultados do modelo univariado no período de testes entre 27/07/2018 e 31/12/2020

6. Links

7. Referências